Visual C++游戏开发案例实战

43.8小时高清多媒体教学视频

王 浩 等编著

全面涵盖从游戏基础知识到游戏项目开发的各种实用技术

实战为王，详细介绍了7个经典游戏项目案例的完整开发过程

◎ **夯实基础**：介绍了游戏类型、集成开发环境、C++语言基础、网络通信基础、游戏中的多媒体处理、项目管理及测试等游戏开发必知必会的基础知识

◎ **案例精讲**：详解五子棋（网络版）、贪吃蛇、俄罗斯方块、连连看、黑白棋、扫雷、推箱子这7个经典游戏项目案例的开发过程，并给出了详细的源代码和注释

◎ **实用性强**：讲解游戏开发的基础知识时给出了60多个实例，讲解游戏项目案例时注重每个项目的设计思路，并将软件工程的思想融入项目开发中

◎ **技巧性强**：讲解过程中穿插了大量的开发技巧、说明及各种注意事项

超值、大容量DVD光盘

◎ **12小时**配套教学视频及本书实例源文件

◎ **20.5小时**Visual C++入门与进阶教学视频

◎ **11.3小时**Visual C++模块与项目开发教学视频

◎ **13个**Visual C++典型模块开发源文件

◎ **3个**Visual C++项目开发案例源文件

◎ **324页**C/C++程序员面试宝典电子书



清华大学出版社

Visual C++游戏开发案例实战

王　浩　等 编著

清华大学出版社
北　京

内 容 简 介

本书是一本介绍电脑游戏项目开发的初中级项目实践教程。书中以 Visual C++为开发平台，结合 7 个游戏开发的经典案例，详细介绍了从游戏开发基础知识到游戏项目开发的实用技术。**配书光盘中提供了专门为本书录制的 12 个小时多媒体教学视频和书中涉及的源代码，另外赠送了大量的进阶开发视频和源代码。**

本书共 16 章，分为 3 篇。其中，第 1～6 章是游戏开发基础篇，讲解游戏项目开发应该具有的准备知识，主要介绍各种游戏类型及常用技术、Visual C++集成开发环境的使用、C++编程语言基础、多媒体处理及项目管理基础知识；第 7～10 章为五子棋游戏案例分讲篇，重点突出其中的项目文档编写、过程控制、网络处理及算法设计；第 11～16 章为其他游戏开发案例篇，详细讲解了贪吃蛇、俄罗斯方块、连连看、黑白棋、扫雷、推箱子等多款游戏的设计、项目文档编写及实例开发。这些游戏不仅涵盖多种游戏经典算法，而且都是精心设计的，富有代表性。每个实例项目的制作步骤都以通俗易懂的语言阐述，并穿插测试与效果演示，比较容易掌握。

本书中的各项目实例之间相互独立，读者可以根据自己的兴趣和需求进行有选择性的学习。本书适合初级或者有一定基础的电脑游戏开发人员，也适合相关院校作为游戏开发的教材使用。

图书在版编目（CIP）数据

Visual C++游戏开发案例实战 / 王浩等编著. —北京：清华大学出版社，2014
ISBN 978-7-302-33762-1

Ⅰ. ①V…　Ⅱ. ①王…　Ⅲ. ①C 语言－程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 211412 号

责任编辑： 夏兆彦
封面设计： 欧振旭
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 30.25　　**字　　数：** 759 千字
版　　次： 2014 年 2 月第 1 版　　**印　　次：** 2014 年 2 月第 1 次印刷
印　　数： 1～0000
定　　价： 00.00 元

产品编号：050592-01

前　言

现在的电脑游戏软件开发都是依靠大量的设计和测试人员共同合作完成的，而如何能够有效控制成本，提高项目开发效率才是重中之重。但在现有大多数的电脑游戏开发教程中，只对游戏中的算法和程序进行了详细讲解，而忽略了现代软件开发最基本的内容，即项目过程管理。本书的目的就是为了让更多的C++语言游戏开发初学者，除了对游戏算法和程序能够充分掌握外，还能够对游戏开发中的项目管理有一个系统、全面的认识。同时为今后参加游戏项目开发打下良好的基础。

笔者结合自己多年的实际项目和团队管理经验精心编写了这本书，目的是让更多的人知道如何编写项目管理文档，同时提高实际项目开发经验，尤其是为电脑游戏开发的新手进入游戏开发行业提供一个项目知识的阶梯。本书也是广大初中级游戏开发人员提高自己的游戏开发水平、完善自己的知识结构、扩展自己的项目知识面的好参谋。

阅读完本书，读者可以有以下收获：

- 让游戏开发初学者能够真正掌握游戏开发的基本知识；
- 建立起基本的项目管理知识，丰富实际项目开发经验；
- 可以单独完成游戏项目管理文档，并能够对用户的需求进行初步分析；
- 可以利用游戏开发的知识，设计简单的VC++游戏程序；
- 可以开发联机的网络游戏，提高游戏开发水平；
- 了解一些完整的项目实例，为以后参加实际项目开发打下一个坚实的基础。

本书特色

本书深入浅出地讲解了各种电脑游戏的基本理论和方法，以及目前流行的各种游戏开发技术和常用的开发工具。本书对游戏开发的基础知识和项目管理的介绍比较详细，而且考虑很多读者在 Windows 编程和开发语言方面还是个新手，所以给出了很多简单的、用C++编程语言来开发的 Windows 程序实例，介绍的比较清晰、易懂。对于一些常见问题，本书给出了套路式解决问题的方法，为初学者学写游戏程序提供了一个练习的途径，并对软件项目管理与软件测试方法进行了详细的讲解，便于读者对这些不熟悉的知识点进行学习。同时，本书采用大量的项目开发实例来对游戏开发过程进行详细讲解，以提高读者的实际项目经验。本书区别于市面上其他的游戏开发类书的特色主要有：

1. 配多媒体语音教学视频光盘

笔者专门为本书录制了 12 个小时高清多媒体教学视频，以便读者更加直观地理解本书内容，提高学习效率。另外，配书光盘中还提供了本书涉及的案例源程序，并赠送了大量的进阶开发视频和源代码，相信对读者的学习会有很大的帮助。

2．由浅入深，循序渐进

本书从游戏开发的基础知识开始讲解，然后从项目开发的角度全面介绍一个完整的五子棋游戏项目案例的开发，最后给出几个各具特色的游戏案例的实现。

3．项目案例丰富、典型

本书中完整实现了五子棋（网络版）、贪吃蛇、俄罗斯方块、连连看、黑白棋、扫雷、推箱子等多款经典游戏项目案例的设计和实现。它们涵盖了多种游戏的经典算法，非常有代表性。

4．代码经典，注释详细

本书详细地讲解了每个项目案例的设计和实现过程，并且给出了详细的核心代码和代码注释，读者只要按照书中的操作步骤和代码解释就可以毫无障碍地阅读本书，并在本书的启发下开发出自己的游戏。

5．注重项目的设计思路

本书并不是简单地给出游戏项目的实现过程，而是在每个项目具体开发前都给出详细的项目分析和设计思路，便于读者从整体上把握项目，提高项目开发水平。

6．注重软件工程思想在实际游戏项目开发中的应用

本书将软件工程的思想渗透到了每个游戏项目开发中，而且每个项目都按照软件工程规范给出了项目开发文档，方便没有项目开发经验的读者了解实际项目开发过程。

7．重点介绍了游戏项目的测试

对于游戏的功能测试是游戏开发中所必须具备的基本知识。因此本书的项目案例都给出了整合测试的相关内容，读者可以在实际开发中随时翻阅，不受基础知识的限制。

8．提供教学 PPT，方便老师教学

本书适合能力培养型的院校和职业学校作为教学用书，所以专门制作了教学 PPT，以方便各院校的老师教学时使用。

本书内容介绍

本书分为 3 篇，共 16 章，从游戏分类讲起，再进一步介绍了各种游戏项目开发需要准备的基础知识。最后结合笔者的经验讲解如何进行实例游戏项目开发，让读者的游戏项目开发水平得以不断的提高。

第 1 篇　游戏开发基础（1～6 章）

本篇主要介绍了游戏开发相关的编程知识。包括电脑游戏的分类及经典作品介绍、常用技术介绍、演示 Visual C++开发 Windows 游戏、C++编程开发语言基础、游戏网络编程

知识简介、简单 Windows 多媒体示例程序开发、游戏项目管理相关内容及文档。

第 2 篇　五子棋游戏案例分讲（7～10 章）

本篇通过分步讲解五子棋游戏开发实例来介绍游戏项目的开发过程。包括五子棋游戏的各种文档的制作、游戏界面的设计、网络通信协议介绍、五子棋游戏核心算法的设计、游戏规则的实现、测试用例文档的编写、相关文档表格的填写及五子棋游戏整合测试的演示。

第 3 篇　高级篇（11～16 章）

本篇主要介绍多个游戏项目开发实例来丰富读者的相关经验。包括贪吃蛇游戏实例开发项目介绍、俄罗斯方块游戏实例开发项目介绍、连连看游戏实例开发项目介绍、黑白棋游戏实例开发项目介绍、扫雷游戏实例开发项目介绍和推箱子实例开发项目介绍。

本书内容由浅入深，理论结合实践，尤其适合初级读者逐步学习和完善自己的知识结构。

本书代码注释约定

- 针对单行代码的注释，都是放在代码的后面；
- 如果单行注释内容过长，与代码无法放置在一行中，则单行注释放在代码的上面；
- 针对函数的注释，统一放在函数开始的{（大括号）右侧并与其他注释上下对齐；
- 针对一段代码的注释，统一放在该段代码的上方，并与其他注释上下对齐。

本书读者对象

- Visual C++游戏开发初学者；
- 没有任何游戏开发学习经验的读者；
- 需要进一步学习游戏核心算法和数据结构的读者；
- 没有参加过项目开发，但想了解项目开发管理的读者；
- 想学习 C++游戏项目开发知识的各大院校计算机专业和非计算机专业的学生；
- 正在学习电脑游戏开发的读者；
- 具备一定编程理论知识，但缺乏实践操作的初级程序人员；
- 从其他语言转向学习 C++游戏程序设计的初中级编程人员。

本书作者

本书由王浩主笔编写。其他参与编写的人员有陈晓建、陈振东、程凯、池建、崔久、崔莎、邓凤霞、邓伟杰、董建中、耿璐、韩红轲、胡超、黄格力、黄缙华、姜晓丽、李学军、刘娣、刘刚、刘宁、刘艳梅、刘志刚、司其军、滕川、王连心、沃怀凯、闫玉宝。

如果你在学习中遇到什么问题，可以通过 bookservice2008@163.com 和我们取得联系。

编著者

目　　录

第 1 篇　游戏开发基础

第 1 章　游戏开发者都应该掌握的知识 ······ 2
1.1　各种游戏类型 ······ 2
1.1.1　角色扮演游戏 ······ 3
1.1.2　动作游戏 ······ 5
1.1.3　冒险游戏 ······ 6
1.1.4　策略游戏 ······ 7
1.1.5　即时战略游戏 ······ 7
1.1.6　格斗游戏 ······ 8
1.1.7　射击游戏 ······ 9
1.1.8　第一人称射击游戏 ······ 9
1.1.9　益智游戏 ······ 9
1.1.10　竞速游戏 ······ 10
1.1.11　体育游戏 ······ 10
1.1.12　养成游戏 ······ 11
1.1.13　模拟游戏 ······ 11
1.1.14　卡片游戏 ······ 12
1.1.15　音乐游戏 ······ 13
1.2　游戏开发技术 ······ 13
1.2.1　图像显示技术 ······ 13
1.2.2　游戏引擎技术 ······ 16
1.2.3　游戏脚本技术 ······ 18
1.3　总结 ······ 19
第 2 章　Visual C++集成开发环境（教学视频：32 分钟） ······ 21
2.1　Visual C++的过去和未来 ······ 21
2.1.1　Visual C++开发工具的由来 ······ 21
2.1.2　Visual C++开发工具的特点 ······ 22
2.2　Visual C++的安装 ······ 22
2.2.1　Visual C++的定制安装 ······ 22
2.2.2　Visual C++的启动 ······ 25
2.3　部署 Visual C++游戏项目 ······ 26
2.3.1　项目中的各种文件的定义 ······ 26

2.3.2 项目文件夹的定义……27
2.4 Windows 的窗体……28
2.4.1 Windows 中的窗体……28
2.4.2 应用程序与窗体的关系……29
2.5 使用 Visual C++开发工具……29
2.5.1 Visual C++开发工具的主界面……29
2.5.2 使用向导创建项目……30
2.5.3 创建一个 Hello World 程序……31
2.5.4 工程文件的配置……34
2.6 总结……35

第 3 章 C++编程语言基础（教学视频：64 分钟）……36

3.1 C++编程语言是什么……36
3.1.1 C++语言的由来……37
3.1.2 C++语言的特点……37
3.2 C++中的各种字符……37
3.2.1 标识符与关键字……38
3.2.2 分隔符与注释符……38
3.3 C++中的常用数据类型……40
3.3.1 整数型数据……40
3.3.2 实数型数据……42
3.3.3 字符型数据……43
3.3.4 布尔型数据……45
3.4 C++中的常量与变量……46
3.4.1 变量的定义……46
3.4.2 常量的定义……47
3.5 C++中的运算符与表达式……48
3.5.1 赋值运算符……48
3.5.2 算术运算符……48
3.5.3 自增与自减运算符……51
3.5.4 复合运算符……52
3.5.5 位运算符……52
3.5.6 关系运算符……54
3.6 C++中的控制语句……55
3.6.1 基本语句……55
3.6.2 条件选择语句……56
3.6.3 循环语句……61
3.7 C++中的数组、指针及引用……64
3.7.1 数组的定义与操作……64
3.7.2 指针的定义与操作……66
3.7.3 引用的定义与操作……69
3.8 函数……70
3.8.1 使用函数的好处……71
3.8.2 函数的定义及声明……71
3.8.3 认识函数的参数……72

3.8.4 函数的调用及返回值……73
3.9 C++的类及其主要函数……75
3.9.1 C++的优点……75
3.9.2 定义 C++类……76
3.9.3 成员变量……77
3.9.4 成员函数……78
3.9.5 构造函数……78
3.9.6 析构函数……79
3.9.7 虚函数……79
3.10 运算符的重载……81
3.11 C++语言的编程规范……83
3.11.1 命名规范……83
3.11.2 格式规范……85
3.11.3 函数规范……86
3.11.4 其他规范……88
3.12 总结……88
3.13 挑战……89

第 4 章 网络通信基础（教学视频：76 分钟）……90

4.1 TCP/IP 简介……90
4.1.1 TCP/IP 整体构架概述……90
4.1.2 TCP/IP 协议的应用……91
4.1.3 TCP/IP 协议的特性……92
4.2 TCP/IP 中的各种协议……93
4.2.1 IP 协议……93
4.2.2 TCP 协议……94
4.2.3 UDP 协议……95
4.3 Socket 简介……96
4.3.1 什么是 Sockets……96
4.3.2 Socket 网络通信模式……96
4.3.3 Socket 的函数……97
4.3.4 Socket 的使用示例……102
4.4 Windows CSockets 类的介绍及使用……104
4.4.1 CAsyncSocket 类和 CSocket 类的介绍……104
4.4.2 阻塞和非阻塞模式……105
4.4.3 类的成员函数介绍……105
4.4.4 CAsyncSocket 和 CSocket 类的编程模型……109
4.5 CAsyncSocket 类综合应用……110
4.5.1 服务器端设计……111
4.5.2 客户端设计……119
4.5.3 综合测试……122
4.6 总结……123
4.7 挑战……123

第 5 章　游戏中的多媒体处理（教学视频：58 分钟）……125
5.1　游戏的多媒体……125
5.1.1　多媒体的概念……125
5.1.2　多媒体技术的特点……126
5.1.3　多媒体能做什么……126
5.2　认识各种多媒体文件……127
5.2.1　Windows 中的文本文件……127
5.2.2　Windows 中的图像文件……128
5.2.3　Windows 中的声音文件……129
5.2.4　Windows 中的视频文件……130
5.3　游戏中图像的显示……132
5.3.1　使用 Pictrue 控件显示图像……132
5.3.2　通过对话框背景显示图像……134
5.3.3　使用 BitBlt()函数动态显示图像……137
5.4　游戏中音乐的播放……141
5.5　游戏中的互动……143
5.5.1　系统对输入设备的处理……143
5.5.2　键盘消息响应……144
5.5.3　鼠标消息响应……148
5.6　两个入门小实例……154
5.6.1　简单的 MP3 播放器……154
5.6.2　简单的图片浏览器……162
5.7　总结……170
5.8　挑战……170
第 6 章　项目管理基础（教学视频：68 分钟）……171
6.1　项目管理……171
6.1.1　项目与项目管理概念……171
6.1.2　项目管理的特点……171
6.1.3　采用项目管理的优势……172
6.2　软件工程与项目管理……173
6.2.1　软件工程的概念……173
6.2.2　软件工程的重要性……174
6.2.3　软件工程管理的流程……176
6.3　需求分析……178
6.3.1　什么是需求分析……178
6.3.2　需求分析的任务和过程……178
6.3.3　需求分析的方法……179
6.3.4　需求分析的 20 条法则……179
6.3.5　深入获得用户的需求……183
6.3.6　可行性分析……183
6.3.7　成本效益分析……184
6.3.8　确定开发环境……185
6.4　项目计划安排……186

6.4.1 项目开发计划的重要性 …… 186
6.4.2 如何制定项目开发计划 …… 186
6.5 总体设计 …… 188
6.5.1 总体设计的概念和目的 …… 188
6.5.2 总体设计的过程 …… 188
6.6 详细设计的工具 …… 188
6.7 软件测试 …… 189
6.7.1 软件测试的目标 …… 189
6.7.2 黑盒与白盒测试 …… 190
6.7.3 软件测试的步骤 …… 190
6.7.4 设计测试方案 …… 191
6.8 软件维护 …… 192
6.8.1 软件维护的概念 …… 192
6.8.2 软件项目的可维护性 …… 192
6.9 总结 …… 193

第 2 篇 五子棋游戏案例分讲

第 7 章 五子棋游戏项目开发的前期工作（教学视频：31 分钟） …… 196
7.1 五子棋游戏的用户需求描述 …… 196
7.2 五子棋游戏的需求说明书 …… 198
7.3 制作五子棋游戏的概要设计文档 …… 199
7.4 五子棋游戏的操作界面设计文档 …… 202
7.5 总结 …… 203

第 8 章 五子棋游戏界面与通信开发详解（教学视频：45 分钟） …… 204
8.1 五子棋游戏的详细设计 …… 204
8.1.1 五子棋游戏详细设计的目标 …… 204
8.1.2 五子棋游戏功能结构及名称定义 …… 204
8.2 网络通信协议类的设计与实现 …… 205
8.2.1 网络通信协议的设计 …… 205
8.2.2 各种数据类型的详细格式 …… 206
8.2.3 网络通信协议的实现 …… 207
8.3 交互界面的设计与实现 …… 207
8.3.1 控制菜单的设计 …… 208
8.3.2 控制菜单的实现 …… 208
8.3.3 网络设置对话框的设计 …… 211
8.3.4 网络设置对话框的实现 …… 212
8.4 总结 …… 214

第 9 章 五子棋游戏的核心算法设计与实现（教学视频：60 分钟） …… 215
9.1 棋盘窗口类的设计与实现 …… 215
9.1.1 棋盘窗口类的设计思想 …… 215
9.1.2 棋盘类的实现 …… 216

9.2 网络交互的设计与实现……222
9.2.1 网络交互的设计思想……222
9.2.2 网络交互的算法实现……223
9.3 游戏规则的设计与实现……225
9.3.1 游戏规则的设计思想……225
9.3.2 游戏规则的算法实现……226
9.4 游戏中主对话框类的实现……230
9.5 总结……235

第 10 章 五子棋游戏整合测试（教学视频：5 分钟）……236
10.1 五子棋游戏的测试用例文档编写……236
10.1.1 引言……236
10.1.2 文档范围……237
10.1.3 使用对象……237
10.1.4 参考文献……237
10.1.5 相关术语与缩略语解释……237
10.1.6 测试项目……237
10.2 根据用例文档进行测试……242
10.2.1 网络连接测试的演示……242
10.2.2 游戏互动测试的演示……244
10.2.3 输赢结果测试的演示……245
10.2.4 禁手功能测试的演示……245
10.2.5 综合测试结果……246
10.3 总结……246

第 3 篇 其他游戏开发案例

第 11 章 贪吃蛇游戏项目开发（教学视频：65 分钟）……248
11.1 贪吃蛇游戏项目的需求分析……248
11.1.1 获得客户需求的语言描述……248
11.1.2 对语言描述进行需求分析……249
11.2 贪吃蛇游戏概要设计……251
11.3 贪吃蛇游戏操作界面及测试用例设计……252
11.3.1 游戏操作界面设计文档……253
11.3.2 测试用例文档……254
11.4 贪吃蛇游戏的详细设计……260
11.4.1 游戏各功能的设计描述……260
11.4.2 游戏各功能的流程图……261
11.5 贪吃蛇游戏界面的实现……262
11.5.1 游戏菜单的实现……262
11.5.2 游戏帮助的实现……266
11.5.3 “英雄榜”的实现……268
11.5.4 游戏背景音乐播放的实现……272

11.6 贪吃蛇游戏核心算法的设计与实现 272
11.6.1 主游戏类的设计 272
11.6.2 主游戏类的实现 273
11.6.3 游戏规则类的设计 280
11.6.4 游戏规则类的实现 280
11.7 贪吃蛇游戏的整合测试 281
11.7.1 游戏等级测试的演示 281
11.7.2 游戏主界面显示功能测试的演示 282
11.7.3 贪吃蛇移动功能测试的演示 282
11.7.4 贪吃蛇游戏规则测试的演示 283
11.7.5 贪吃蛇游戏分数统计测试的演示 283
11.8 总结 284

第 12 章 俄罗斯方块游戏项目开发（教学视频：41 分钟） 285

12.1 俄罗斯方块游戏项目的需求分析 285
12.1.1 获得客户需求的语言描述 285
12.1.2 对语言描述进行需求分析 286
12.2 俄罗斯方块游戏概要设计 288
12.3 俄罗斯方块游戏操作界面及测试用例设计 289
12.3.1 游戏操作界面设计文档 290
12.3.2 测试用例文档 291
12.4 俄罗斯方块游戏的详细设计 297
12.4.1 游戏各功能的设计描述 298
12.4.2 游戏的各功能流程图 299
12.5 俄罗斯方块游戏的界面实现 299
12.5.1 游戏菜单的实现 299
12.5.2 游戏帮助对话框的实现 302
12.5.3 游戏英雄榜对话框的实现 304
12.5.4 游戏播放背景音乐的实现 307
12.5.5 游戏等级设置对话框的实现 308
12.6 俄罗斯方块游戏的核心算法设计与实现 310
12.6.1 主游戏类的设计 310
12.6.2 主游戏类的实现 311
12.6.3 游戏规则类的设计 322
12.6.4 游戏规则类的实现 322
12.7 俄罗斯方块游戏的整合测试 324
12.7.1 主菜单和界面显示功能测试的演示 324
12.7.2 游戏等级选择功能测试的演示 324
12.7.3 方块移动功能测试的演示 325
12.7.4 游戏规则功能测试的演示 325
12.7.5 游戏帮助功能测试的演示 326
12.7.6 游戏计分功能测试的演示 326
12.8 总结 327

第 13 章 连连看游戏项目开发（教学视频：49 分钟）……328
13.1 连连看游戏项目的需求分析……328
13.1.1 获得客户需求的语言描述……328
13.1.2 对语言描述进行需求分析……329
13.2 连连看游戏项目概要设计……331
13.3 连连看游戏操作界面及测试用例设计……332
13.3.1 游戏操作界面设计文档……332
13.3.2 测试用例文档……333
13.4 连连看游戏的详细设计……339
13.4.1 游戏各功能的设计描述……339
13.4.2 游戏各功能流程图……340
13.5 连连看游戏的界面实现……341
13.5.1 游戏菜单的实现……341
13.5.2 游戏帮助对话框的实现……343
13.5.3 游戏英雄榜对话框的实现……345
13.5.4 游戏播放背景音乐的实现……347
13.6 连连看游戏的核心算法设计与实现……348
13.6.1 主对话框类的设计……348
13.6.2 主对话框类的实现……349
13.6.3 棋子类的设计……357
13.6.4 棋子类的实现……358
13.7 连连看游戏的整合测试……368
13.7.1 主菜单和界面显示功能测试的演示……368
13.7.2 消除相同棋子功能测试的演示……368
13.7.3 游戏升级功能测试的演示……369
13.7.4 消除提示功能测试的演示……370
13.7.5 游戏帮助功能测试的演示……370
13.7.6 棋子换盘功能测试的演示……370
13.8 总结……371
第 14 章 黑白棋游戏项目开发（教学视频：51 分钟）……372
14.1 黑白棋游戏项目的需求分析……372
14.1.1 获得客户需求的语言描述……372
14.1.2 对语言描述进行需求分析……373
14.2 黑白棋游戏概要设计……375
14.3 黑白棋游戏操作界面及测试用例设计……376
14.3.1 游戏操作界面设计文档……376
14.3.2 测试用例文档……377
14.4 黑白棋游戏的界面实现……382
14.4.1 游戏菜单的实现……382
14.4.2 游戏帮助对话框的实现……385
14.4.3 游戏播放背景音乐的实现……386
14.5 黑白棋游戏的核心算法设计与实现……387
14.5.1 棋盘窗口类的设计……387

14.5.2 棋盘窗口类的实现 ······ 388
14.5.3 人工智能模块的设计 ······ 393
14.5.4 人工智能模块的实现 ······ 395
14.6 黑白棋游戏的整合测试 ······ 406
14.6.1 主菜单和界面显示功能测试的演示 ······ 406
14.6.2 悔棋功能测试的演示 ······ 406
14.6.3 棋子动画翻转功能测试的演示 ······ 407
14.6.4 游戏胜负判断功能测试的演示 ······ 408
14.6.5 游戏帮助功能测试的演示 ······ 408
14.7 总结 ······ 409

第 15 章 扫雷游戏项目开发（教学视频：52 分钟） ······ 410

15.1 扫雷游戏项目的需求分析 ······ 410
15.1.1 获得客户需求的语言描述 ······ 410
15.1.2 对语言描述进行需求分析 ······ 411
15.2 扫雷游戏概要设计 ······ 412
15.3 扫雷游戏操作界面及测试用例设计 ······ 414
15.3.1 游戏操作界面设计文档 ······ 414
15.3.2 测试用例文档 ······ 415
15.4 扫雷游戏的界面实现 ······ 419
15.4.1 游戏菜单的实现 ······ 419
15.4.2 游戏帮助对话框的实现 ······ 421
15.4.3 游戏英雄榜对话框的实现 ······ 423
15.4.4 游戏播放背景音乐的实现 ······ 425
15.5 扫雷游戏的核心算法设计与实现 ······ 426
15.5.1 新游戏处理模块的设计与实现 ······ 426
15.5.2 地雷格子模块的设计与实现 ······ 428
15.5.3 游戏规则模块的设计与实现 ······ 430
15.5.4 游戏绘图模块的设计与实现 ······ 432
15.5.5 玩家输入模块的设计与实现 ······ 433
15.6 扫雷游戏的整合测试 ······ 435
15.6.1 主菜单和界面显示功能的测试演示 ······ 435
15.6.2 鼠标输入功能的测试演示 ······ 436
15.6.3 标示指定格子功能的测试演示 ······ 436
15.6.4 游戏胜负判断功能的测试演示 ······ 437
15.6.5 游戏帮助功能的测试演示 ······ 437
15.7 总结 ······ 438

第 16 章 推箱子游戏项目开发（教学视频：44 分钟） ······ 439

16.1 推箱子游戏项目的需求分析 ······ 439
16.1.1 获得客户需求的语言描述 ······ 439
16.1.2 对语言描述进行需求分析 ······ 440
16.2 推箱子游戏概要设计 ······ 441
16.3 推箱子游戏操作界面及测试用例设计 ······ 443
16.3.1 游戏操作界面设计文档 ······ 444

16.3.2 测试用例文档 ······ 445
16.4 推箱子游戏的界面实现 ······ 450
16.4.1 游戏菜单的实现 ······ 450
16.4.2 游戏帮助对话框的实现 ······ 452
16.4.3 游戏关口选择对话框的实现 ······ 454
16.4.4 游戏播放背景音乐的实现 ······ 456
16.5 推箱子游戏的核心算法设计与实现 ······ 456
16.5.1 地图文件读取模块的设计与实现 ······ 457
16.5.2 地图绘制模块的设计与实现 ······ 457
16.5.3 键盘操作模块的设计与实现 ······ 460
16.5.4 游戏规则模块的设计与实现 ······ 463
16.5.5 主对话框的设计与实现 ······ 464
16.6 推箱子游戏的整合测试 ······ 465
16.6.1 主菜单和界面显示功能的测试演示 ······ 465
16.6.2 键盘操作功能的测试演示 ······ 466
16.6.3 箱子放置到指定位置时变色显示功能的测试演示 ······ 466
16.6.4 支持地图扩展功能的测试演示 ······ 467
16.6.5 游戏胜负判断功能的测试演示 ······ 467
16.6.6 游戏帮助功能的测试演示 ······ 468
16.7 总结 ······ 468

第 1 篇　游戏开发基础

因为计算机可以存储各种信息，会按人们事先设计的程序自动完成计算、控制等许多工作，而且还可以模仿人脑的许多功能，代替人脑的某些思维活动，所以计算机又被称做电脑。近年，随着计算机技术的发展，越来越多的家庭拥有了自己的电脑。使用电脑来玩游戏，已经成为很平常的事情。

但亲爱的读者朋友们，在玩别人开发的游戏时，是否想过自己亲手开发出一套游戏呢？作为游戏迷，笔者也有过这样的想法。

为了使读者快速了解并掌握游戏开发的相关编程知识，本篇的第 1 章将讲解游戏的大体分类，包括动作游戏、策略游戏、角色扮演游戏、体育游戏、模拟游戏、即时战略游戏、冒险游戏和益智游戏。除此之外，还加入了游戏中涉及的常用技术介绍，让读者对游戏与游戏设计有一个整体概念。

考虑到很多读者在 Windows 编程方面还是个新手，所以笔者特意在本篇的第 2 章，向读者讲解如何用 Visual C++ 6.0 来开发 Windows 游戏，并且给出了简单的例子。在学习如何使用 Visual C++ 6.0 之后，读者可以阅读第 3 章，在这一章中会学习到关于 C++编程开发语言的基础知识。目前，网络游戏非常流行，在第 4 章中就有关于游戏开发中使用网络通信的相关内容。在这里读者将学习到游戏网络编程开发方面的知识。而在本书的第 5 章中，读者就可以根据笔者提供的案例和代码，开发出简单的 Windows 音乐和图像等多媒体程序。第 6 章是游戏项目管理，主要讲解游戏项目管理相关的内容及各种文档的介绍。

第 1 章　游戏开发者都应该掌握的知识

本章采用传统游戏分类的方法，即按游戏类型分类（这里所指的游戏全部是指电脑游戏，以后不作单独说明），对各种不同游戏类型进行简单介绍，并把游戏开发中常用的技术和游戏设计的基本概念进行简单讲解。

本章主要涉及的内容如下：

- 各种不同种类的游戏介绍：让读者了解各类游戏的特点及主要代表作。
- 游戏相关技术的介绍：让读者了解游戏中各种常用的技术及其发展过程。

1.1　各种游戏类型

在游戏世界里，玩家通常会遇到许多不同类型的游戏，这些游戏因为其操作方法、使用人数、人物视角、游戏目的及制作风格特点的不同，可以有很多不同的游戏分类。但这种分类目前尚无统一标准。

说明：现在大部分的游戏分类方法，是按照游戏内容的主题进行分类。

本节中将根据笔者自己的认识对这些游戏进行类别划分，如下所示。

- 角色扮演游戏：Role-Playing Game，简称 RPG。
- 冒险游戏：Adventure Game，简称 AVG。
- 动作游戏：Action Game，简称 ACT。
- 策略游戏：Strategy Game，简称 SLG。
- 格斗游戏：Fighting Game，简称 FTG。
- 即时战略游戏：Real-Time Strategy Game，简称 RTS。
- 射击游戏：Shooting Game，简称 STG。
- 第一人称射击游戏：First Personal Shooting Game，简称 FPS。
- 益智游戏：Puzzle Game，简称 PZL。
- 体育游戏：Sports Game，简称 SPT。
- 竞速游戏：Racing Game，简称 RAC。
- 模拟游戏：Simulating Game，简称 SIM。
- 养成游戏：Education Game，简称 EDU。
- 卡片游戏：Card Game，简称 AG。
- 音乐游戏：Music Game，简称 MSC。

除对将这些游戏分类外，笔者还将简单介绍这些分类中包括的经典代表作品。

1.1.1　角色扮演游戏

角色扮演类游戏，绝对是大多数游戏玩家接触得最多的游戏之一。其主要特点是：由玩家负责扮演一个或多个角色；角色如同真实人物一般成长，有完整的故事情节；并且有关于人物本身的很多属性供玩家调整。

角色扮演类游戏按其游戏风格又可划分为日式 RPG 和美式 RPG。它们的主要区别在于文化背景和战斗方式不同。

（1）日式风格的重点在于故事情节本身，即把整个游戏当成一个长长的故事。而玩家的任务在于展开故事的时间线，而且日式 RPG 多采用回合制或半即时制战斗。其中有两款经典的日本游戏作为代表：“勇者斗恶龙”与“最终幻想”系列。而国产的角色扮演游戏，大多数也可以归属于日式风格，例如“仙剑奇侠传”和“剑侠”系列，如图 1.1 所示。

（2）美式风格的重点在于不同的世界观，在游戏中增加了人工智能等要素。玩家的主要任务是控制主角人物做出选择，如同真实生活中的社会一样。美式风格的 RPG 经典游戏的代表作品是“魔法门”（如图 1.2 所示）、“创世纪”和“博德之门”系列。其特点是力图营造一个虚拟的真实社会，提供极高的玩家选择自由度，而剧情方面则明显淡化，以史诗类型的剧情为主。



图 1.1　仙剑奇侠传游戏截图



图 1.2　魔法门游戏截图

注意：不管哪种风格的游戏，其内容才是吸引玩家的主要手段。

角色扮演类游戏还可以根据战斗进行方式分类，例如：

1．动作角色扮演游戏（Action Role-Playing Game，简称 ARPG）

ARPG 的战斗方式为即时动作形，其不存在切换画面的过程，是直接在地图上进行即时的战斗。而传统 RPG 战斗方式一般是由地图画面切换至战斗画面，然后进行回合制或半即时、即时的战斗。其代表作品是“暗黑破坏神”系列，如图 1.3 所示。

2．战略角色扮演游戏（Strategy Role-Playing Game，简称 SRPG）

SRPG 的战斗方式是以回合制战斗为基础，以战棋模式进行战斗的 RPG。此类游戏不仅具有了策略游戏（后面将对其进行介绍）的战斗要素，同时又具备 RPG 的重要元素人物与情节。其会淡化战斗以外的游戏操作，整个游戏流程基本就是“剧情——战斗——准

备——剧情”的循环。此类游戏以日本、国产居多。其代表作品有“炎龙骑士团”、“三国志英杰传”和“火焰之纹章”系列（如图 1.4 所示）。



图 1.3　暗黑破坏神游戏截图



图 1.4　三国志英杰传和火焰之纹章游戏截图

3．电子化桌上角色扮演游戏（Computerized Role-Playing Game，简称 CRPG）

在介绍 CRPG 之前，还需要向读者介绍一下其前身“桌上角色扮演游戏（TRPG）”，在 TRPG 中除了几个玩家以外，还必须存在一个游戏的“主持人”，其负责设定游戏世界、维护游戏规则、制作游戏的场景，安排剧情、操控非玩家人物及怪物等。而在电子化的这类游戏中，则由电脑扮演主持人的角色。这些游戏一般会忠实地还原 TRPG 的规则。

CRPG 就是在这个基础上发展而来的，所以其战斗系统既非 ARPG 的完全即时，亦非 SRPG 的完全回合化。这类游戏的代表作有“博德之门”（如图 1.5 所示）、“无冬之夜”和“冰风谷”系列。其都是构建于桌面角色游戏“龙与地下城规则”之上的 CRPG。



图 1.5　博德之门游戏截图

4．大型多人线上角色扮演游戏（Massively Multiplayer Online Role-Playing Games，简称 MMORPG）

MMORPG 是现在最为流行的、通常所谓的“网络游戏”，实际上是线上游戏的一种。基本模式是设置一系列的“任务”供玩家完成，玩家从中获取各种“报酬”以提升主角的等级、各种能力、技能等属性。通常玩家可以自由选择各种属性升级的方向。

而 RPG 中的剧情要素在 MMORPG 中被极度淡化，因此可以说 MMORPG 是美式 RPG 游戏系统方面的升级版本。MMORPG 通常提供传统 RPG 所不具备的社交系统，如设置行会、工会等供玩家加入和交流，也可以设置好友、师徒及婚姻等多种人际关系。另一方面，

MMORPG 由于是基于互联网的，其社交的范围很大，所以能举行远比传统 RPG 要多的游戏活动（如攻城、物品交易等），游戏中的商业氛围也远比单机游戏浓厚。其中的代表作品有“传奇世界”、“魔兽”、“A3”等（如图 1.6 所示）。



图 1.6　魔兽和传奇世界游戏截图

说明：正是由于网络游戏具有高交互性能和精致的界面，所以其在国内的市场占有率达到 70%以上。

1.1.2　动作游戏

动作游戏是由玩家控制游戏人物消灭敌人以过关的游戏，没有具体的故事情节和人物成长。现在电脑上的大多数动作游戏都是以早期的街机游戏和动作游戏为基础发展而来的。其纯粹是面向普通玩家的，以纯娱乐休闲为目的。

该游戏角色的操作方式简单，可以让玩家快速融入游戏之中，并通过角色的各种动作来考验玩家的反应能力。随着游戏难度的增加，会给玩家带来相当大的成就感。这是属于“大众化”游戏，也是较受欢迎的游戏种类，其中一小部分还包含简单的解谜内容。

该类游戏的代表作品有“合金弹头”、“波斯王子”、“魂斗罗”、“三国志”系列（如图 1.7 所示）。这些代表游戏中有一些含有射击操作，如“合金弹头”系列。不过由于此类游戏中更注重对游戏角色身体的控制而非射击技术，所以仍属于 ACT 分类。

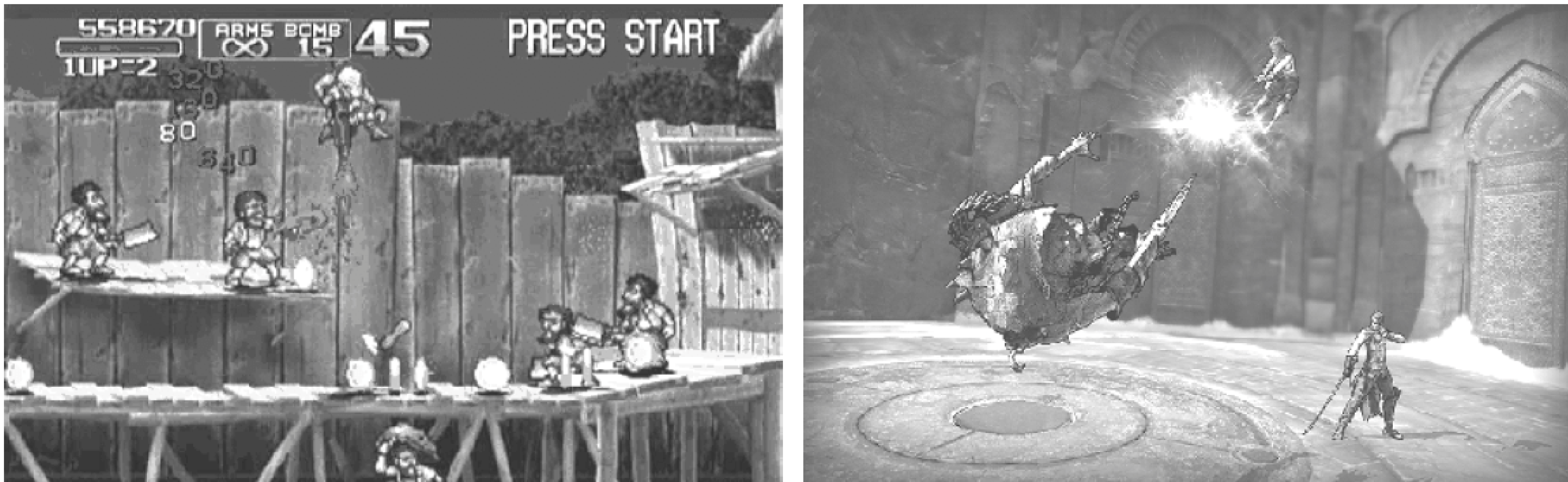


图 1.7　波斯王子和合金弹头游戏截图

1.1.3　冒险游戏

冒险游戏，其与前面介绍的 RPG 游戏一样存在主角，由玩家控制游戏主角进行虚拟冒险的游戏。AVG 的特点是故事情节往往是以完成一个任务或解开某些迷题的形式出现的，在游戏过程中强调解开谜题的重要性，减少了较多的战斗场景，同时角色能力的成长较之 RPG 也相对简单。

说明：正是由于冒险游戏的特点，其又可细分为动作类冒险游戏、解谜类冒险游戏及文字类冒险游戏。

1．动作类冒险游戏

这类游戏以第三人称的视角进行游戏，一般包含较多的格斗或者射击等动作要素，主要考验玩家的游戏操控能力。该类游戏的战斗场景不算太多，但也有人物的等级或者技能等属性的成长，同时还包含少量的解迷元素，其中最具有代表性的作品是“生化危机”、“古墓丽影”和“恐龙危机”系列（如图 1.8 所示）。



图 1.8　古墓丽影和生化危机游戏截图

2．解谜冒险游戏

这类游戏以第一或第三人称的视角进行游戏，主要是解谜冒险活动。其基本抛弃了战斗场景和人物操控能力，纯粹依靠解谜拉动剧情的发展。通常仅需要移动场景、选择对话，以及解开谜题。该类游戏难度系数较大，其中经典代表作品有“神秘岛”系列，如图 1.9 所示。

3．文字类冒险游戏

此类游戏以第一人称的视角进行游戏，以剧情主导，在游戏时，相当于在看一本小说。游戏中通常还会引入诸如 CG、音乐等收集要素。一般的模式是通过主角在剧情中的选择产生不同的结局。

其中最主流的题材是侦探和恋爱类两种。如图 1.10 所示，侦探文字类的代表作是“侦探神宫寺三郎”和“逆转裁判”系列，恋爱类的代表作是《秋之回忆》系列。这里需要注意的是，恋爱文字类与后面将要介绍的恋爱模拟游戏是有所区别的。恋爱文字的冒险游戏

在剧情的深度与广度上远远胜于后者。



图 1.9　神秘岛游戏截图

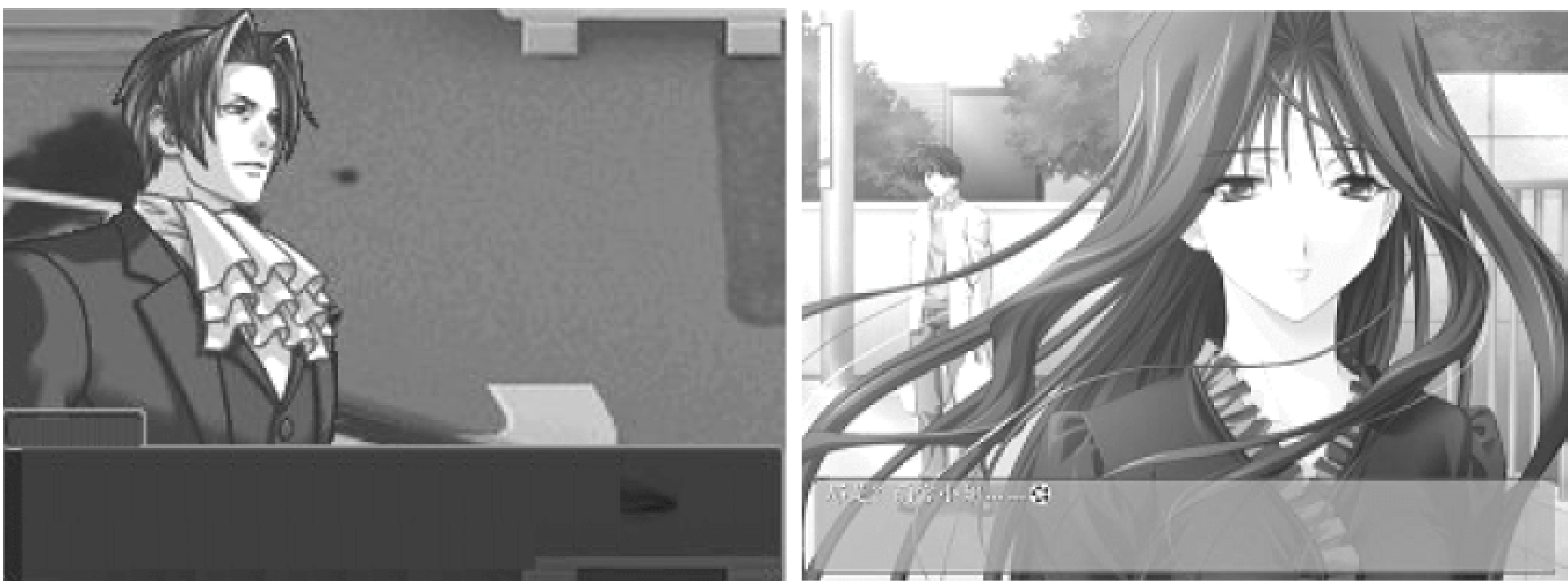


图 1.10　逆转裁判和秋之回忆游戏截图

1.1.4　策略游戏

策略游戏，大多数是以战争故事题材为背景，其要求玩家“拥有”做出决策的能力。在策略游戏中，除了需要玩家的熟能生巧外，头脑的灵活性往往也会对游戏的结果产生至关重要的影响，当然其中也包含很多运气成分。

大部分策略游戏最大的特点就在于：如何充分调动玩家的智慧来配置游戏中的各种资源达到游戏目标，或统一全国，或开拓外星殖民地，或取得局部战役的胜利。策略游戏可以细分为如下两种。

1．回合制

这类 SLG 中，玩家只能在自己的回合时间内，才能进行各种动作，如发展经济、经营军备等。其中经典的代表作品有“三国志”、“魔法门之英雄无敌”（如图 1.11 所示），以及“信长之野望”和“文明”系列。前面提到过的 SRPG，就是借用 SLG 战斗模式的 RPG。



图 1.11　三国志和魔法门之英雄无敌游戏截图

2．即时制

这类 SLG 中，玩家可以和其他玩家及电脑同时进行各种战略活动，进行实时的较量。这也是后面将要讲解的即时战略游戏（RTS）的前身。其中经典的代表作品有“帝国时代”、“沙丘魔堡”、“红色警戒”、“星际争霸”系列等。

1.1.5　即时战略游戏

即时战略游戏是由 SLG 发展而来。本来属于策略游戏 SLG 的一个分支，但由于其在

世界上的迅速风靡，使之发展成了一个单独的类型，知名度甚至超过了 SLG。其各方面与 SLG 相仿，但在经济、兵种方面往往比较简单，讲究情报、资源、操作等优势。

注意： 很多时候在划分策略游戏时，即时战略游戏也是被划分在策略游戏分类之中的。这里只是为了讲解需要才单独划分出来。

该类游戏经典的代表作品有“帝国时代”、“星际争霸”、“魔兽争霸”系列等。而其中“帝国时代”更是以历史文化演变为背景，把历史故事加入到游戏之中，任务玩法多变，场景细腻丰富，满足了不同玩家的需求，征服了整个游戏市场。

后来，从即时战略游戏之上又衍生出了所谓的“即时战术游戏”，多以控制一个小队完成任务的方式，突出游戏中战术的作用，其以“盟军敢死队”系列为经典代表作品，如图 1.12 所示。



图 1.12　帝国时代与盟军敢死队游戏截图

1.1.6　格斗游戏

格斗游戏，是动作游戏的某种简化与强化形式，其主要由玩家操纵各种角色与电脑或另一玩家所控制的角色进行格斗的游戏。此类游戏谈不上有什么剧情，最多有个简单的场景设定或背景展示。人物的操控较易掌握，但操作难度较大，主要依靠玩家迅速的判断和微操作取胜。

按其显示技术，该类游戏可再细分为 2D 和 3D 格斗两种，如图 1.13 所示。其中，2D 格斗游戏有著名的“街霸”、“侍魂”、“拳皇”系列等。3D 格斗游戏有“铁拳”、“高达格斗”系列等。



图 1.13　拳皇和铁拳游戏截图

1.1.7　射击游戏

射击游戏，这里所说的射击类，并非是类似“VR 战警”的模拟射击（枪战）游戏，而是指纯粹的空战类射击游戏。一般由玩家控制各种飞行物（主要是飞机）通过射击消灭敌人、击中目标，完成任务最后过关的游戏。

根据操作的角色和可控性又可细分为纵轴射击、横轴射击两类。纵轴射击的代表作如“雷电”系列，横轴射击代表作如“沙罗曼蛇”系列，如图 1.14 所示。

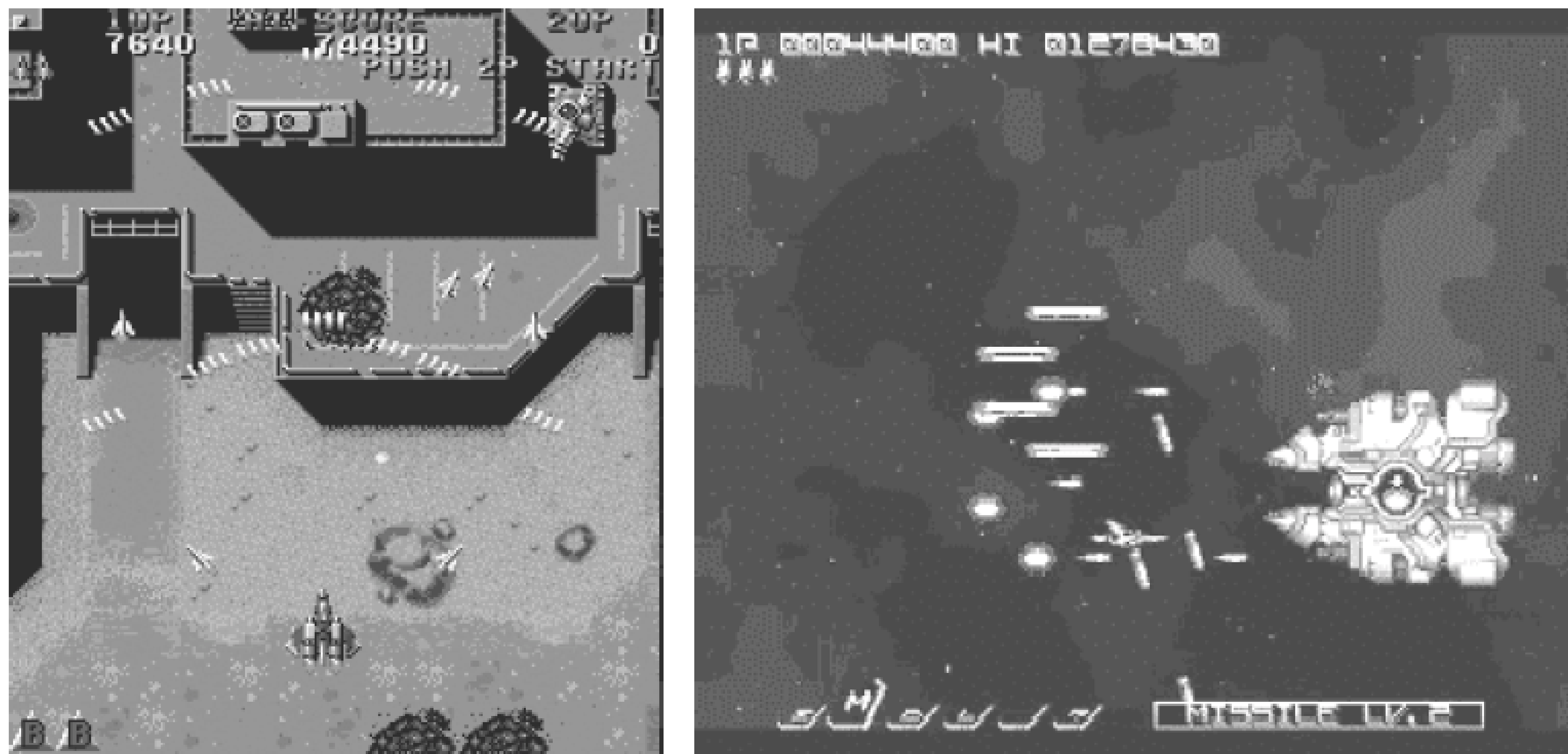


图 1.14　雷电和沙罗曼蛇游戏截图

1.1.8　第一人称射击游戏

第一人称射击游戏，其属于动作游戏的一个分支，但和 RTS 一样，由于其在世界上的迅速风靡，使之发展成了一个单独的类型。这类游戏限定了玩家必须以第一人称的视角来处理游戏中所有相关画面，以射击作为战斗的第一要素。正因为这类游戏通常都是以 3D 画面呈现游戏的界面，所以很大程度上引领着游戏引擎技术的发展潮流。

其中经典的代表作品有 DOOM、“雷神之锤”、“虚幻”、“半条命”、CS 系列等，如图 1.15 所示。此外与此类型相仿的，还存在着第三人称射击游戏，但数量稀少，著名作品如“英雄本色”系列。



图 1.15　CS 和雷神之锤游戏截图

1.1.9　益智游戏

益智类游戏，Puzzle 的原意是指以前用来培养儿童智力的拼图游戏，引申为各类有趣的益智游戏，也可以叫做休闲游戏。其特点是通常规则简单、操作简易，目的是让玩家得到休息与放松。一般也将“中国象棋”、“围棋”和各种牌类游戏划分到益智游戏中。最经

典的休闲类游戏当属“俄罗斯方块”、“五子棋”等游戏，如图 1.16 所示。

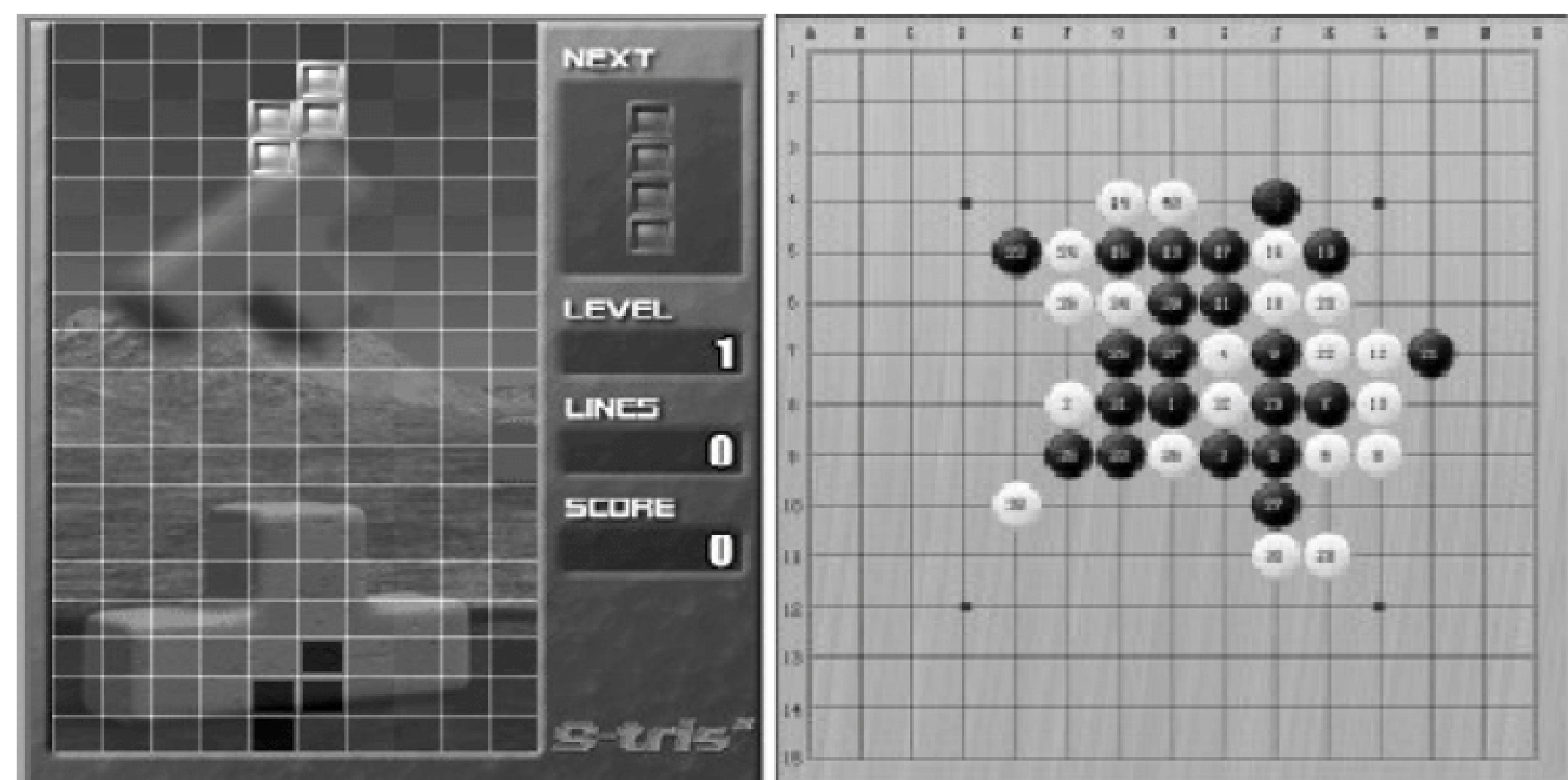


图 1.16　俄罗斯方块和五子棋游戏截图

1.1.10　竞速游戏

竞速游戏，其是在电脑上模拟各类竞速运动的游戏，通常是在比赛场景下进行，非常讲究图像音效技术，往往是代表电脑游戏的尖端技术。惊险刺激，真实感强，题材多以赛车为主，所以深受车迷玩家的喜爱。其中代表作品有“极品飞车”、“山脊赛车”、“摩托英豪”等系列，如图 1.17 所示。目前，RAC 内涵越来越丰富，也出现了另一些其他模式的竞速游戏，如赛艇，赛马等。



图 1.17　极品飞车和摩托英豪游戏截图

竞速类游戏本应划归体育类游戏（SPT），但由于其流行程度大致与体育类游戏相当，而且游戏方式大异于各类以双方对垒为主的体育类游戏，故单独成为一类。目前这类游戏还称为 Driving Game。

1.1.11　体育游戏

体育游戏，是指在电脑上模拟各类竞技体育运动的游戏。从篮球、足球到雪上运动，甚至高尔夫球运动都被引入到游戏中。其花样繁多，模拟度高，所以广受欢迎。

这类游戏也让玩家有着另外一种很奇妙的感情，例如，许多玩家可能会因为支持真实世界中的某个团队而爱上这类体育游戏，甚至为让所支持的队伍打败其他队伍，不惜日夜苦练支持的队伍，以此满足自己的好胜心。

体育游戏的代表作品有 FIFA、NBA Live、“实况足球”系列等，如图 1.18 所示。现在市面上几乎所有体育运动游戏均可以在 EA Sports 的作品中见到。



图 1.18　FIFA 和 NBA 游戏截图

1.1.12　养成游戏

养成类游戏，顾名思义，就是玩家以养成为主要目的的游戏，多数以强化自身能力、培养人际关系，或者增强感情、提高他人能力为题材。有些人将养成类游戏与恋爱类游戏等同。实际上并不存在单一的“恋爱类游戏”这一类型，恋爱游戏多以 AVG 和 EDU 的形式存在。

目前，这类游戏主要是为男性玩家服务的（可以训练追求的技术等），也有个别是面向女性玩家的。其代表作品有“心跳回忆”、“明星志愿”系列，如图 1.19 所示。



图 1.19　心跳回忆和明星志愿游戏截图

1.1.13　模拟游戏

模拟游戏，就是指在电脑上模仿出现实中的各种事物或者事件发生的游戏。其最大的特点就是模拟力求完美，游戏操作指令较为复杂，重点突出物理原则及给玩家的真实感受，让玩家在玩游戏中获得置身其中的真实感，并可以让玩家在特定状况中完成真实世界中难以完成任务。根据模拟对象的不同，其又可以细分为模拟经营类和飞行模拟类。其各自特点如下所述。

1．经营模拟类（又称模拟策略游戏）

玩者须妥善规划游戏中金钱的运用以达到成功目标，其中涉及资金调度与投资策略。取材自真实世界，所以此类游戏有着各式各样的模拟题材，例如，大到一个国家小到一间农场的建设与管理，模拟的对象很广泛。其代表作品有“模拟城市”、“铁路大亨”、“模拟人生”系列，如图 1.20 所示。

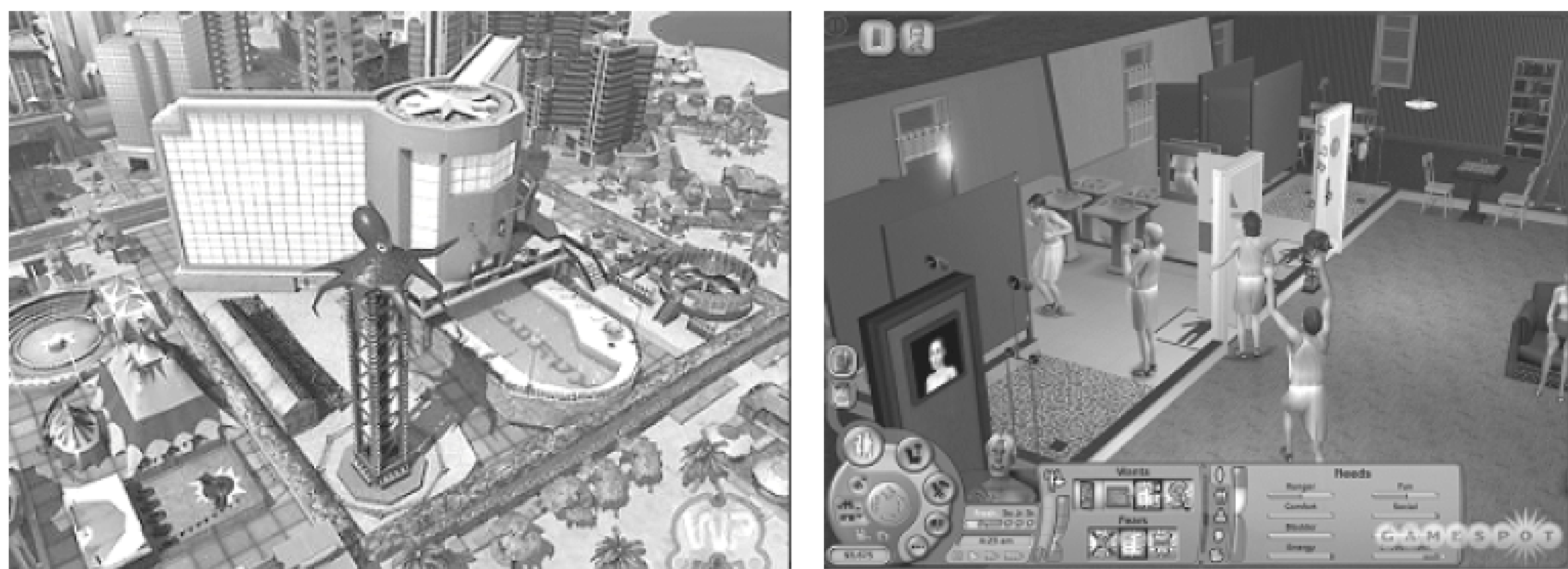


图 1.20　模拟城市和模拟人生游戏截图

2．飞行模拟游戏（Real-Simulation Game，简称 RSG）

该类游戏以现实世界为基础，以真实性取胜，追求拟真，达到身临其境的感觉。所以设计上较重视物体的数学及物理反应。简单地说，在这类游戏中，一颗铅球从半空中落下，绝不会像羽毛那样随风飘动，任何物体的移动都要符合物理学上的原理。其代表作品有“王牌空战”、“模拟飞行 2002”系列等，如图 1.21 所示。另外，还有一些模拟其他对象的游戏也可归为 STG，比如模拟潜艇的《猎杀潜航》，模拟坦克的《钢铁雄师》等。



图 1.21　王牌空战和模拟飞行 2002 游戏截图

1.1.14　卡片游戏

卡片游戏，又称卡牌对战游戏，是指玩家操纵角色通过卡片战斗模式来进行的游戏。丰富的卡片种类使得游戏富于多变化性，给玩家无限的乐趣，代表作品有“信长的野望”、“游戏王”系列，也包括卡片网游“武侠 Online”，从广意上说“王国之心”也可以归于此类，如图 1.22 所示。



图 1.22　信长的野望游戏截图

1.1.15　音乐游戏

音乐游戏，培养玩家音乐敏感性，增强音乐感知的游戏。伴随美妙的音乐，有的要求玩家翩翩起舞，有的要求玩家手指体操。大多数都是以某种小游戏的形式配合玩家的节奏感来进行。其中的代表作品有“跳舞机”、“太鼓达人”，还有目前的人气网游“劲乐团”，如图 1.23 所示。但要注意，目前的热门网游“劲舞团”虽然具备音乐要素，但其音乐与游戏进行几乎完全无关，所以不能算作 MSC，只能划入 PZL。



图 1.23　劲乐团和跳舞机游戏截图

1.2　游戏开发技术

毫无疑问，现代游戏开发需要涉及许多方面的技术。就像很多相关书籍中所提到的那样：游戏开发是同时满足美术、音乐、声音，以及核心编程的完美结合等众多要求而且还要使这些要求之间不会相互发生冲突的技术。在游戏开发中需要艺术和科学同时存在，也只有这里才能实现创造力和边缘科学的真正结合。笔者将在本节中，把游戏开发中涉及的一部分常用技术进行简单地讲解和介绍，让读者们有一个初步的了解。

1.2.1　图像显示技术

电脑游戏中的图像、视频的显示和输出，都是依靠电脑中的显卡来实现的。显卡接受电脑内部的数字信息并把其转化为肉眼可见的显示信号。绝大多数的电脑里，显卡将数字信息转换为能够在显示器上显示的模拟信息。当然现在采用 DVI 接口的液晶显示器，其数据仍然保持数字信号形式。

根据游戏中图像呈像方式的不同，一般可以把图像的显示技术分为 2D 和 3D 显示两种。但对于玩家而言，2D 技术和 3D 技术只是显示数据的方式而已，玩家都是通过二维的平面显示器来观看的。

所以在真正开始讲解图像显示技术之前，还是先来认识一下图像中的 2D 坐标系统，一种是平面几何中的 *XY* 坐标系统，另一种是电脑显示屏上的 *XY* 坐标系统。

在平面几何的 *XY* 坐标系统中，*X* 坐标轴代表的是象限中的横坐标轴，坐标值由左向右线性递增；*Y* 坐标轴代表的是象限中的纵坐标轴，从标值由下向上线性递增，如图 1.24 所示。

如果在一个很近的距离内观察电脑的显示屏，就会看到屏幕上所有的物体都是由一个一个的点所构成。这些点被称为像素（Pixel），每一个像素都由红、绿、蓝三色组合而成。一般的屏幕分辨率为 1024×768 或者图像的分辨率为 1024×768，这都是指的屏幕或者画面在宽度（*X* 轴）上可以显示 1024 个点，而高度方向（*Y* 轴）上可以显示 768 个点。在显示屏上的平面坐标系统如图 1.25 所示。

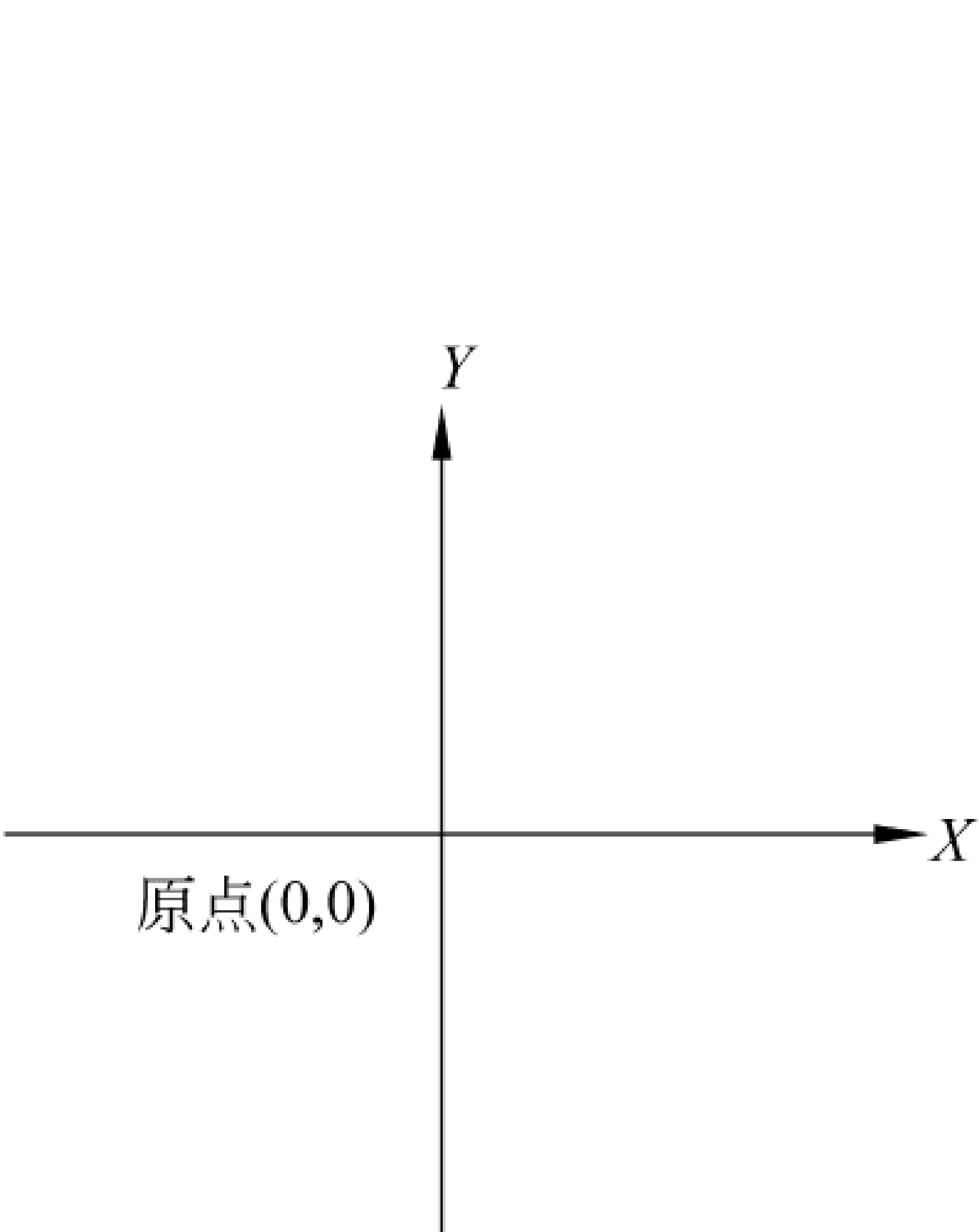


图 1.24　平面几何中的 *XY* 坐标系统



图 1.25　显示屏上 *XY* 坐标系统

在显示屏上，所有平面坐标都只有正坐标，而没有负坐标，而且都是以屏幕左上角为原点的，*X* 轴仍然是从左向右线性递增，而 *Y* 轴则是由上向下线性递增。虽然其也可以接受负值，但如果 *X* 或者 *Y* 坐标为负值，那么这些对应的点就位于屏幕外，而不会显示在屏幕中，也没有实际意义。

屏幕中坐标系统的大小由显示器的分辨率来决定，一般经常使用的有 640×480、800×600、1024×768 及 1280×720 等，其都是屏幕上对应的坐标点。例如，800×600 就是说明 *X* 坐标轴上有 800 个像素点，*Y* 坐标轴上有 600 个像素点。

1. 2D和3D图像显示技术的概念

图像的显示都是在二维平面坐标系中，是平面的，没有立体感，只有 *x* 轴和 *y* 轴。所有图形元素是以平面图片的形式制作和显示的，这就是 2D 图像显示技术。如图 1.26 左图所示，就是一个 2D 图像的显示。

2D 图像中的动画是以一帧的形式预先存在的。这些图形元素最终都会以复杂的联系方式在游戏中进行调用，而实现游戏世界中丰富的内容。

而图像的显示都是在三维立体标系中，物体是真实占有空间的，即有立体感。而且可以以任何人的视点（摄像机）来任意移动并改变视角。这就是 3D 图像显示技术。如图 1.26 右图所示，就是一个 3D 图像的显示。

3D 技术把游戏世界中的每个物体看做一个个立体的对象，由若干个几何多边体构成。为了显示对象，在文件中存储的是对对象的描述语句。3D 图像在显示时，程序通过对这些语句的解释来实时地合成一个物体。通过若干个立体几何和平面几何公式的实时计算，玩家在平面的显示器上还能以任意的角度来观看 3D 物体。即使仅仅是图形显示上的变化，在 3D 引擎下世界构成的任何事物也要以 3D 世界观来对待。

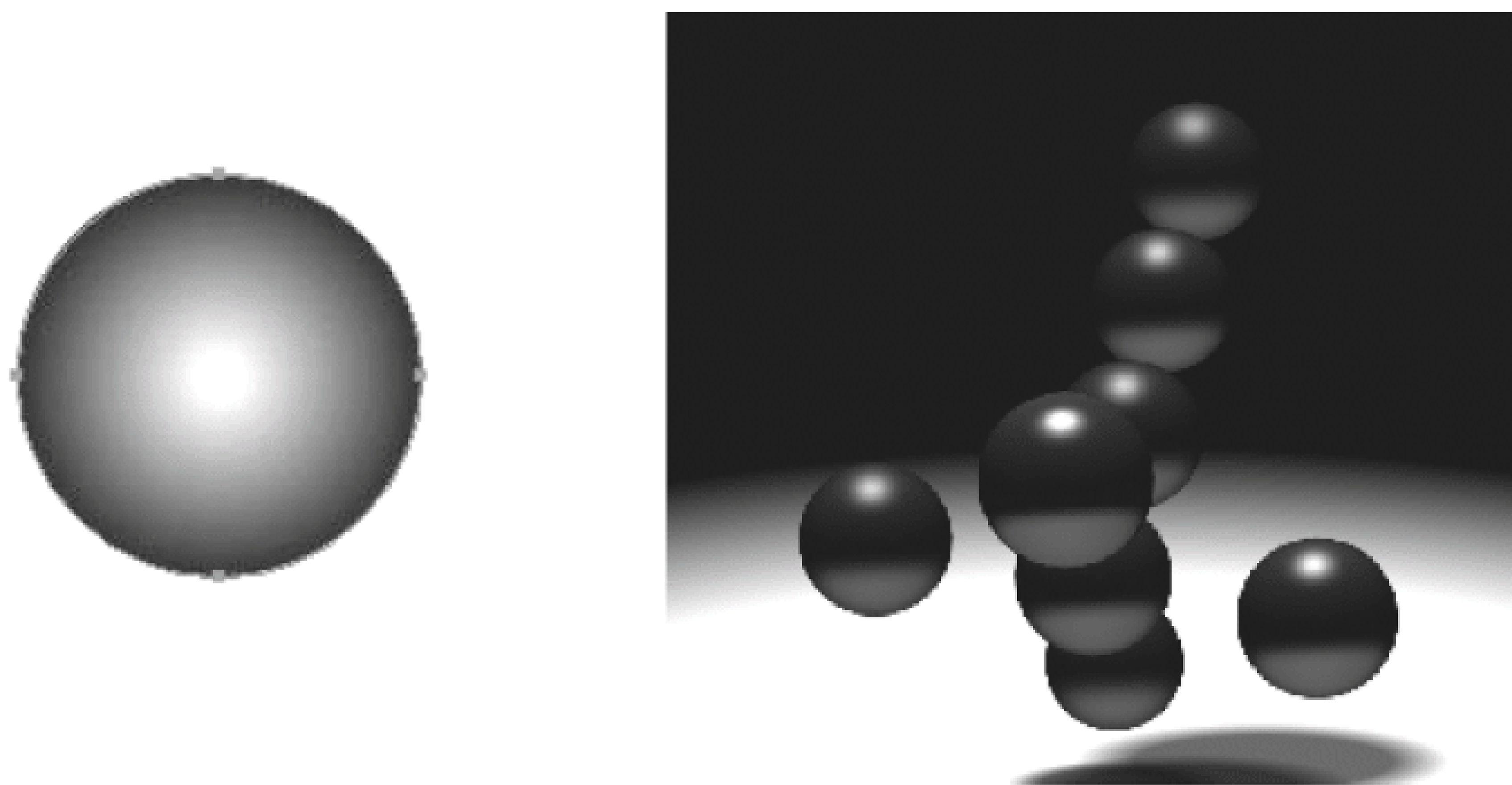


图 1.26　2D 和 3D 显示例图

2．2D 和 3D 图像显示技术的比较分析

3D 技术相对于 2D 技术的优点有如下几个方面。

- 3D 技术在三个重要的方面显得非常灵活，首先就是表现在你眼前的世界 3D 游戏能够让玩家以任意的角度来观看世界，并可以让玩家在其中以任意角度观察。显然通过 2D 技术是实现不了的，因为表现的结果有上万种可能性。
- 3D 技术在动画制作方面有独特的优势。如果用 2D 技术来实现，那么在视角的转换上就会非常受限，而且存储这些动画会超过所能承受的最大空间。3D 动画能够很轻松地越过这些困难，只要先给 3D 对象定义变形和运动的规律。实际运行时，程序就会让 3D 对象按照预定的规律来运动，形成 3D 动画。但是在现有的游戏中，还没有很好地发挥这一优势。
- 3D 对象易于修改，因为其由若干个多边形组合而成，可以像搭积木一样重新搭建。这些多边形可以像橡皮泥一样任意地揉捏，来符合最终的要求。而 2D 图像是由手工绘制而成，修改非常不便，一些大的改动往往导致已有的工作成果作废，需要重新绘制。

例如，有时需要表现游戏中的人物在四个方向上行走时的动作，那么 2D 就必须预先画好起码四幅图像，游戏运行时分别从不同的文件中调用行走时的图像，来表现行走动作。如果这时人物需要能在八个方向上行走，那么工作量就会增加一倍，在制作一些动画时，工作量更是急剧地增长。这时，使用 3D 技术就能省下不少的时间。

但 2D 技术也不是一无可取，2D 技术相对于 3D 技术，也有其优点。

- 2D 的图像能够画得很精致，可以把一些细节完美地表现出来。3D 虽然也能通过增加多边体来更细致地表现对象，但与 2D 图像所能达到的最高水准还是相差一段距离。如果你的游戏不需要诸如旋转视角等功能的话，那么采用 2D 技术会给你的玩家带来更好的视觉享受。
- 2D 技术在屏幕上显示和处理都很快，因为所有的图像都已经预先处理好了，设计者所要做的只是把其从文件中调出来并显示到屏幕上。利用显卡来做这些工作是绰绰有余的。在实时的游戏中，你能够轻易地获得每秒几十帧的显示速度，这就为处理器节省了大量的时间，使其有充裕的时间来做显示以外的其他工作。例如，对即时战略游戏来说，电脑控制方必须具有一定的人工智能才有挑战性，所以其游戏算法就比较复杂，会占用较多的处理器时间去计算过程。现在 3D 加速卡的速度已经变得越来越快了，但如果要模拟一个真正的 3D 世界，这点速度还远远不够。

所以至少在未来的几年内，2D 图像的显示速度还将占有较大的优势。

- ❑ 使用 2D 的图像技术来做一个显示引擎很容易，但对于 3D 图像来说，制作显示引擎就困难得多了。随着技术的普及，这一优势会渐渐消失。
- ❑ 一般的 3D 游戏具有较高的操作自由度，对有经验的玩家来说是没问题的。但对一些新手来说，自由度越高就感到越难以控制。2D 游戏一般具有较少的操作自由度，新手容易在设计者的引导下一步步地进行下去。

总之，作为游戏设计者，需要决定采用哪种技术。如果设计的游戏需要比较自由的空间，倾向于动作化，并且有强大的技术力量，那么 3D 引擎将是最好的选择。如果只是想制作具有精美图像的游戏，不需要太高的自由度，那么 2D 技术是最好的选择。

注意：事实上，3D 技术与 2D 技术并不存在什么孰优孰劣的问题，只存在着哪种技术能更好地为玩家和设计者服务的问题。

1.2.2 游戏引擎技术

在游戏开发中也有类似赛车的“引擎”，其相当于游戏的核心，决定着整个游戏的速度的稳定性。玩家在游戏中体验到的剧情、关卡、美工、音乐、操作等内容都是由游戏的引擎直接控制的。其把游戏中所有的元素捆绑在一起，并指挥这些元素有序地工作。

引擎就是“用于控制所有游戏功能的主程序，从计算物体的碰撞物理系统和图像的显示，到接受玩家的输入，以及按照正确的音量输出声音等”。简单地说，游戏引擎就是指通过游戏设计的模型构建一个“平台”，能够方便地支持游戏开发的后续工作。根据不同的游戏类型，可以分为 STG 引擎、RPG 引擎、ACT 引擎等类型。同样，也可以根据采用显示技术的不同，分为 2D 引擎和 3D 引擎。

除此以外，引擎还有一个重要的职责就是负责玩家与电脑游戏之间的沟通，处理来自键盘、鼠标、摇杆和其他外设的信号。如果游戏支持联网特性的话，网络代码也会被集成在引擎中，用于管理客户端与服务器之间的通信。

其实，说了这么多，引擎就是经过不断的进化，由多个子系统共同构成的复杂系统，游戏中的引擎是整个游戏的框架，在框架打好后，编剧、美工、关卡设计师、建模师、动画师只需要往里填充内容就可以了。所以，一般游戏开发中，引擎的开发占全部开发总量的 50%以上。

下面笔者将介绍一下整个游戏引擎的进化过程。

1992 年，Wolfenstein 3D 引擎，其作者是大名鼎鼎的约翰·卡马克。这个引擎开创了第一人称射击游戏的先河，更重要的是，其在 *X* 轴和 *Y* 轴的基础上增加了一根 *Z* 轴，在由宽度和高度构成的平面上增加了一个向前和向后的纵深空间，这根 *Z* 轴对那些看惯了 2D 游戏的玩家造成的巨大冲击可想而知，如图 1.27 所示。

1993 年，Doom 引擎，其同样是出自 id Software 公司，在技术上大大超越了前面的 Wolfenstein 3D 引擎，因为 Wolfenstein 3D 引擎的所有路径之间的角度都是直角，是说玩家只能笔直地前进或后退，而 Doom 引擎中路径之间的角度可以为任意，而且墙壁的厚度也可以为任意的，如图 1.28 所示。



图 1.27　Wolfenstein 3D 游戏截图



图 1.28　Doom 游戏截图

1995 年，Quake 引擎，也是出自 id Software 公司，是当时第一款完全支持多边形模型、动画和粒子特效的真正意义上的 3D 引擎，而不是和 Doom 一样的 2.5D 引擎。此外 Quake 引擎还是连线游戏的始作俑者，把网络游戏带入大众的视野之中，促成了电子竞技产业的发展，如图 1.29 所示。

1997 年，Quake II 引擎，其充分地利用 3D 加速和 OpenGL 技术，在图像和网络方面与前作（Quake）相比有了质的飞跃，也确定了 id Software 公司在 3D 引擎市场上的霸主地位，如图 1.30 所示。



图 1.29　Quake 游戏截图



图 1.30　Quake II 游戏截图

1997 年，Unreal 引擎，是使用最广的一款引擎，其出自 Epic 公司。游戏中的许多特效即便在今天看来依然很出色。而且其应用范围不限于游戏制作，还涵盖了教育、建筑等其他领域。经过不断的更新，至今依然活跃在游戏市场上，丝毫没有显出老迈的迹象，如图 1.31 所示。



图 1.31　Unreal 游戏截图

1998 年，Half-Life 引擎，采用了 Quake 和 Quake II 引擎的混合体，加入了脚本序列技术和人工智能技术，使得游戏中敌人的行为比以往游戏中的敌人更加“聪明”和“狡诈”，如图 1.32 所示。

1998 年，Dark 引擎，出自 Looking Glass 工作室，虽然其在图像方面比不上 Quake II 引擎，但是

在人工智能方面却是真正取得突破的游戏引擎。例如，游戏中的敌人会懂得根据声音辨认玩家角色的方位；能够分辨出不同地面上的脚步声。而且在不同的光照环境下有不同的视力，发现同伴的尸体后会进入警戒状态，还会针对角色的行动做出各种合理的反应等难得见到的人工智能行为。

2000 年，Quake III 引擎，是在 Quake II 出色的图像引擎基础上加入了更多的网络成分，如图 1.33 所示。



图 1.32　Half-Life 游戏截图



图 1.33　Quake III 游戏截图

2000 年，Unreal Tournament 引擎，其在画面处理上的改变与 Quake III 引擎相比并不多，但是在互联网模式上，其不仅提供了死亡竞赛模式，还提供了团队合作等多种激烈火爆的对战模式，而且 Unreal Tournament 引擎不仅可以应用在动作射击游戏中，还可以为大型多人游戏、即时策略游戏和角色扮演游戏提供强有力的 3D 支持，如图 1.34 所示。

2004 年，Unreal Tournament 2 引擎，在画面上又比前一代有了更高的进步，对于网络的支持也更强大和高效，如图 1.35 所示。



图 1.34　Unreal Tournament 游戏截图

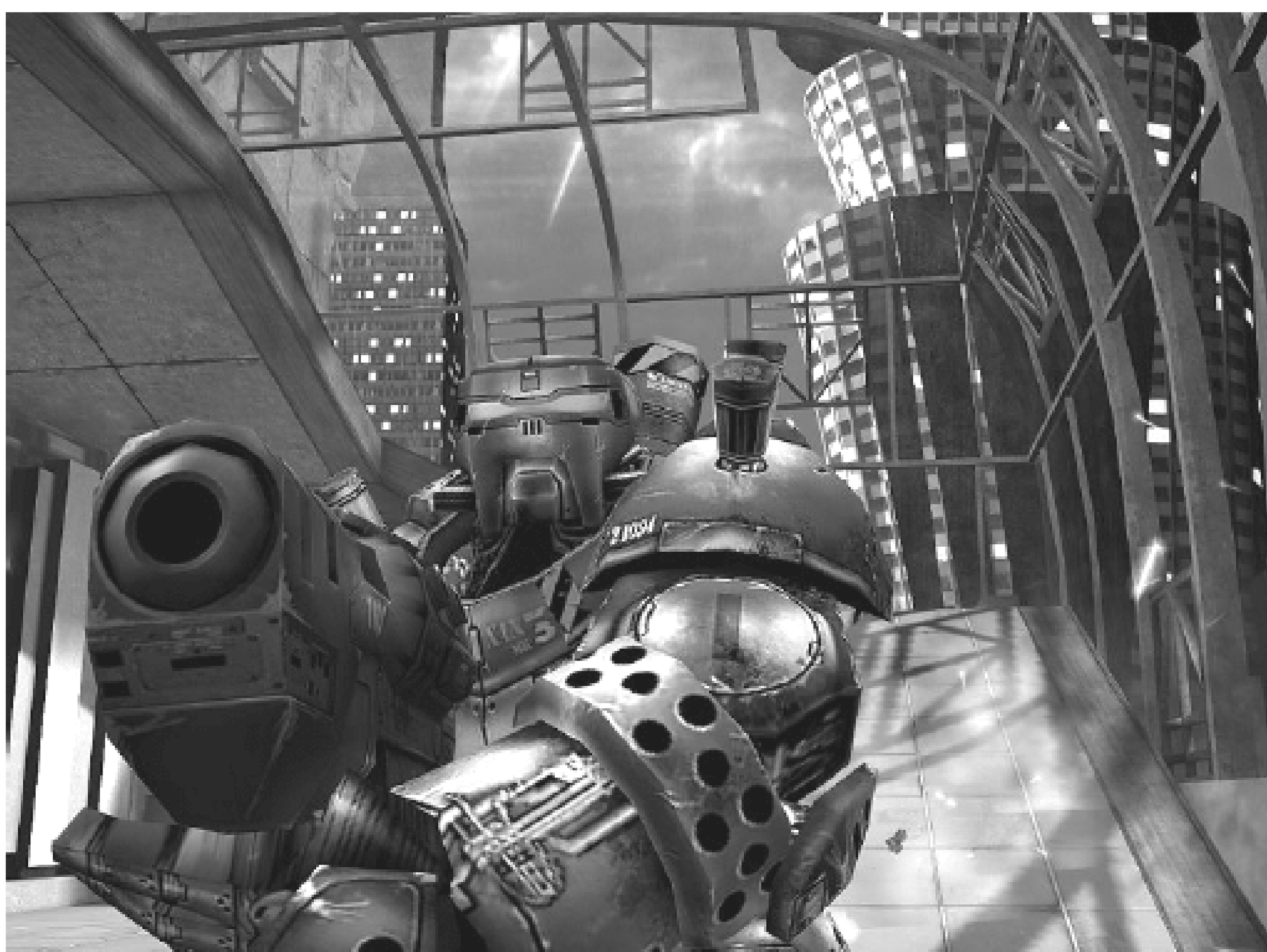


图 1.35　Unreal Tournament 2 游戏截图

1.2.3　游戏脚本技术

所谓游戏脚本技术，就是指游戏中的角色和事物都支持通过脚本来进行控制和描述。什么是脚本呢？脚本就像是运行在游戏内部的小程序，可以使用一个普通的文本编辑器来编写，然后使用一个编译器来编译通过，最后提供一个编译后文件。其工作原理和其他的

普通程序一样，但又与普通应用程序不同。

二者之间的区别在于普通的可执行文件在创建完成后，除了重新编译外，就不能再被扩展了，所以普通可执行文件也被称为硬编码文件。而脚本生成的可执行文件，并不像其他普通的可执行文件一样可以在电脑上直接运行，而是在执行时，进行实时的代码翻译工作后再执行。

简单地说，脚本就是一条条文字命令，这些文字命令是可以看到的，并由系统的一个解释器将其一条条地翻译成机器可以识别的指令，并按程序的顺序执行。如果读者使用过DOS 操作系统，那么就应该体会更深，脚本相当于批处理文件。

在游戏中实现脚本技术，最根本的原因就是要避免硬编码。如果将游戏内容和游戏引擎相分离，设计者就可以在不需要重复编译整个工程的情况下调整、测试和修改游戏运行的机制和特性。

游戏脚本技术也同时使得游戏在编译、打包和封装以后还能够很容易地进行扩展，如图 1.36 所示。游戏中的逻辑也可以被视为模块化的内容，允许其像图形和声音一样灵活和可交互。所有的修改和扩展部分都可以被玩家下载并能够快速地被游戏所识别。有了这样一套系统，设计出来的游戏就可以无限制地进行扩展了（当然，只要人们能够编写出新的脚本和内容）。

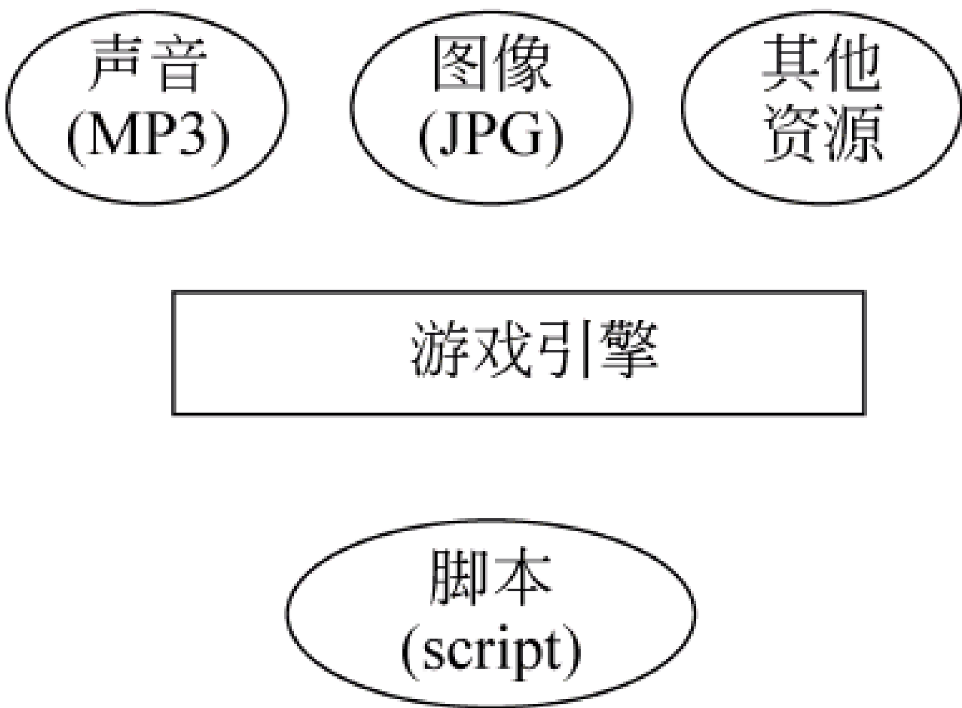


图 1.36　游戏内容和游戏引擎分离

游戏的脚本技术可以让脚本和游戏进行通信，而游戏也可以作出应答。其可以像加载图形、声音一样加载。因为游戏引擎和游戏内容的完全分离，也使得在没有一行具体游戏代码的情况下游戏引擎仍然可以编译运行。

因此，游戏玩家使用的真正的游戏完全可以由脚本和其他一些媒体，如声音和图形组成。这就意味着当游戏玩家购买游戏软件的时候，实际购买了两个分离的部分：一个编译过的游戏引擎和一系列可以对游戏功能进行扩展的脚本。当有新脚本时，玩家又可以再去下载，而无需重新安装整个游戏。

总之，在游戏中添加了脚本技术，使得游戏设计者在游戏的早期只需要关注游戏引擎的开发，而不用注意游戏角色、事务流程、剧本开发、AI（人工智能）设计等细节性工作，从而提高了游戏引擎的开发效率。

1.3　总　　结

通过这一章的学习，是不是觉得这是一个很好的开始？仅仅在一章之内，你已经迅速了解了游戏开发的世界，知道了各种游戏的分类，有了基本的概念，大概了解了各分类的特点，并对这些分类中的代表作品也有认识。除此以外，对游戏中的图像显示、游戏引擎和脚本技术也建立了基本概念。

如果你刚开始学习这些东西，那么你就可以先自我鼓励一下然后继续前进。在本章中，读者需要了解的内容如下：

- 15 种游戏分类的划分及其各自的特点。

- 图像显示技术中 2D 技术与 3D 技术的特点和区别。
- 游戏引擎技术的概念及其发展。
- 游戏脚本技术的优点及目的。

在第 2 章中，读者将会学习真正的程序开发工具——Visual C++的使用。因此，接着向下阅读吧，因为只有先掌握基本开发工具的使用，才能够学习和理解那些更加深入的内容。在下一章节中，读者将了解到：

- Visual C++开发工具的特点。
- Visual C++的安装。
- 如何部署 Visual C++游戏项目。
- 对话框的区别。
- 使用 Visual C++开发工具。

最后，再次提醒读者朋友，电脑显示屏上的坐标系都是以左上角为原点的正坐标系统，其 *Y* 轴是从上向下线性递增，而不是平面几何中的坐标系统，而且负值是没有意义的。

第 2 章　Visual C++集成开发环境

读者通过前一章的学习，已经大体了解了游戏如何分类及其常用的技术。有了这些基本概念，才能对游戏开发有一个整体的认识。

俗话说“工欲善其事，必先利其器”。本章中将介绍开发游戏的工具——Visual C++，其是微软公司开发的面向 Windows 程序设计的一整套开发环境，能使开发 Windows 应用程序变得更加容易，其中的 GDI+对于游戏开发有很高的效率。下面就开始学习如何安装及使用 Visual C++。

本章主要涉及的内容如下：

- Visual C++简介：了解 Visual C++的发展历史及特点。
- Visual C++的安装：掌握好如何安装 Visual C++，对于以后的开发工作大有裨益。
- 部署 Visual C++游戏项目：知道一个完整的游戏项目的文件如何安排及存放。
- 窗体的分类：了解 Windows 中各种不同的窗体。
- 使用 Visual C++开发工具：掌握 Visual C++开发工具创建工程的方法及配置。

2.1　Visual C++的过去和未来

随着科学技术的不断发展，人们对电脑软件的需求也在不断地增加，软件项目日趋庞大，软件开发技术日渐成熟，一个功能强大且易用的开发工具逐渐成为开发人员驰骋沙场的利器。

2.1.1　Visual C++开发工具的由来

在全世界，Windows 操作系统被广泛使用，Windows 平台下的软件开发也成为软件开发人员的必备技能。Visual Studio 系列开发系统一直是在 Windows 操作系统下进行软件开发的一套非常实用的工具集。其可以用来开发多种 Windows 下的软件项目，包括 Windows 应用程序、动态链接库、Windows 服务、Office 集成开发、数据库项目开发等。

Visual C++ 6.0 就是微软公司开发的面向 Windows 程序设计的一整套开发环境——Visual Studio 中的一种开发工具。同时也是微软公司面向 Windows 操作系统（包括 Windows NT、Windows 2000、Windows XP 等）出品的可视化的快速开发工具的产品。

Visual C++是以可视化技术为基础，以 C++为主要编程语言，集成众多工具的开发利器。其操作简单，界面和功能设计符合程序员的开发习惯，同时配合使用微软官方开发的帮助文档 MSDN，可以给设计和开发工作带来更大的便利。在一般的游戏设计中，Visual C++也是非常好的开发工具。

技巧：在使用 Visual C++开发工具时，查阅 MSDN 文档可以提高开发效率。

2.1.2　Visual C++开发工具的特点

Visual C++提供的 MFC 类库，是一个很大的、扩展了的 C++类层次结构，其能使开发 Windows 应用程序变得更加容易。而且 MFC 在整个 Windows 家族中都是兼容的，也就是说，无论是 Windows 98 还是 Windows XP，所使用的 MFC 是兼容的。每当新的 Windows 版本出现时，MFC 也会得到修改以便使旧的编译器和代码能在新的系统中工作。MFC 也会得到扩展，添加新的特性、变得更加容易建立应用程序。

使用 MFC 的最大优点是其做了所有最难做的事。MFC 中包含了成千上万行正确、优化和功能强大的 Windows 代码。其调用的很多成员函数可以帮助你完成自己可能很难完成的工作。一般性的界面开发工作也可以全部交给其来完成，用户就只需要在这些基础上做出自己想要实现的功能即可。与其他开发工具相比，Visual C++完成 Windows 图形界面的程序所花费的时间要少得多。

由于 MFC 编程方法充分利用了面向对象技术的优点，使得我们编程时极少需要关心对象方法的实现细节，同时类库中的各种对象的强大功能足以完成程序中绝大部分所需的功能，这使得应用程序中程序员所需要编写的代码大为减少，有力地保证了程序的良好性和可调试性。

近几年经过微软公司不断地完善，Visual C++在开发速度、程序执行效率、程序文件大小与系统的集成性方面都有极大的提高，这使得其不但适应开发一般的应用软件，还可以开发数据库应用软件，同时更适合图形图像和文件压缩等算法编程。

另外由于其是微软公司推出的，因此也能够较好地与 Windows 平台接合，深入操作系统的内部和底层，实现高级程序设计要求，提高程序的运行效率。

有一句关于 Visual C++的话在网络上流传甚广，即“偷懒的人学 VB（即 Visual Basic），聪明的人学 Delphi，真正的程序员学 VC（即 Visual C++）”，也充分说明了 VC 的特点。

2.2　Visual C++的安装

本节的主要内容是讲解如何配置和安装 Visual C++开发环境。如果你是一个 Windows 程序开发的初学者，那么首先就需要知道如何安装 Visual C++开发环境。但如果你是一个有程序开发经验的读者，会配置开发环境，那么可跳过本节，学习后面的章节。

2.2.1　Visual C++的定制安装

安装和配置 Visual C++开发环境的步骤如下所述。

（1）进入光盘目录，并启动安装程序，如图 2.1 所示，其包含了 Visual Studio 开发工具的全部内容。

（2）双击其中的 Setup.exe 文件图标，就可以启动 Visual Studio 6.0 的安装程序，如图 2.2 所示。在欢迎界面中，包含了 Visual Studio 6.0 的简单介绍和安装程序的使用方法，

如果想详细了解，可以单击其中的 View Readme 按钮进行了解。在这里，直接单击 Next 按钮，进入下一步安装过程。

图 2.1 光盘目录结构图

图 2.2 安装程序主界面

（3）在这一步中，会出现安装许可说明，其中规定了用户使用 Visual Studio 6.0 的权力、义务及需要注意的事项，如图 2.3 所示。选中 I accept the agreement 单选按钮（表示同意许可说明中的条款），并单击 Next 按钮，进入下一步安装过程。

（4）在这一步中，当用户输入正确的产品序列号、用户的姓名和用户公司名称后（这里可以随便输入），如图 2.4 所示。然后单击 Next 按钮，可以把这些相关信息记录到电脑中，并进入下一步安装过程。

图 2.3 许可证协议对话框

图 2.4 注册登记对话框

（5）在这一步中，是设置安装选项，一般情况下选择 Enterprise Setup Options 选项区域中的 Custom 单选按钮，表示采用定制模式，如图 2.5 所示。单击 Next 按钮，进入下一步安装过程。

（6）在这一步中，可以设置安装文件存放的路径及文件夹，当你指定了目录后，其会告知当前的硬盘空间是否足够，当不足时，就需要指定其他盘进行安装。默认情况还是放在系统盘下，如图 2.6 所示。单击 Next 按钮进入下一步安装过程。



图 2.5　安装配置选项对话框

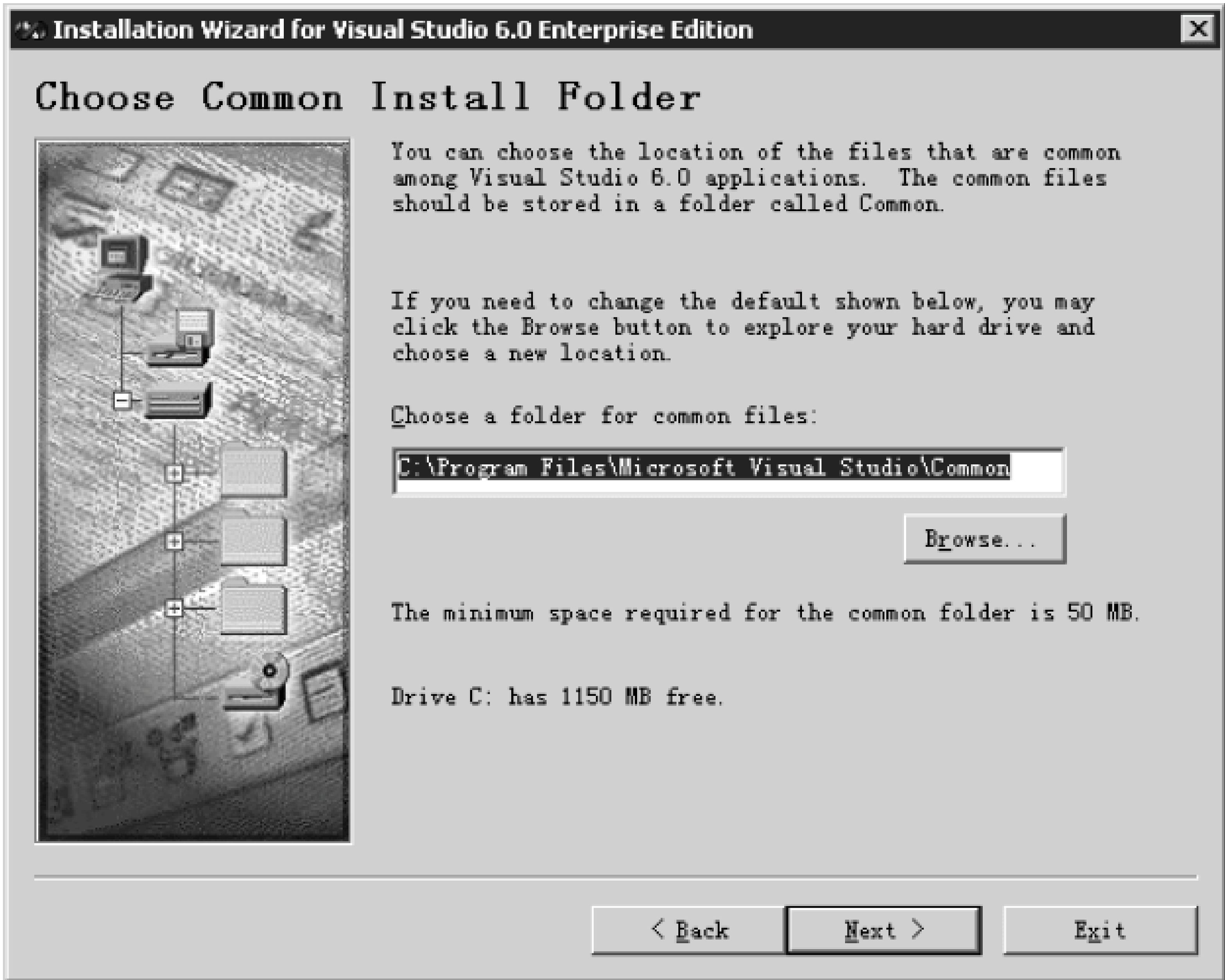


图 2.6　指定安装文件存放路径对话框

（7）在这一步中，主要是一些注意事项和提示信息，如图 2.7 所示，直接单击 Continue 按钮进入下一步安装过程。如果想终止安装，可以单击 Exit Setup 按钮。

（8）在这一步中，同前面一样，也是一些提示信息，不过在这个对话框中包含了前面输入的序列号，在这里给出提示，如图 2.8 所示。直接单击 OK 按钮进入下一步安装过程。



图 2.7　提示注意对话框



图 2.8　序列号提示对话框

（9）在这一步中，安装程序会检查系统，如果发现系统中以前安装有 VSS 6.0（源代码版本管理软件）以前的版本，会提醒用户进行升级，如图 2.9 所示，如果用户觉得不需要升级，可以单击 No 按钮进入下一步安装过程。

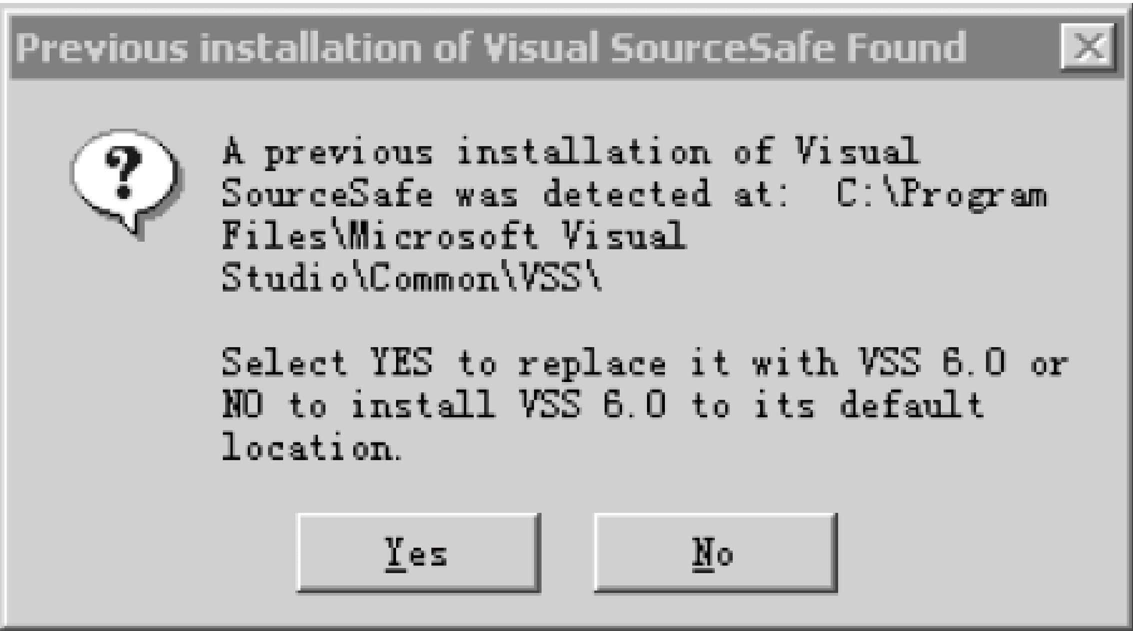


图 2.9　VSS 6.0 更新对话框

（10）前面检查完系统文件后，在这一步中，就可以定制需要的 Visual Studio 开发工具。因为在本书中，是以 Visual C++为主来讲解，所以我们只需要按图 2.10 中所示，把相应的选项（Microsoft Visual C++ 6.0）选中即可，把其他选项取消掉。单击 Continue 按钮进入正式安装过程，如图 2.11 所示。

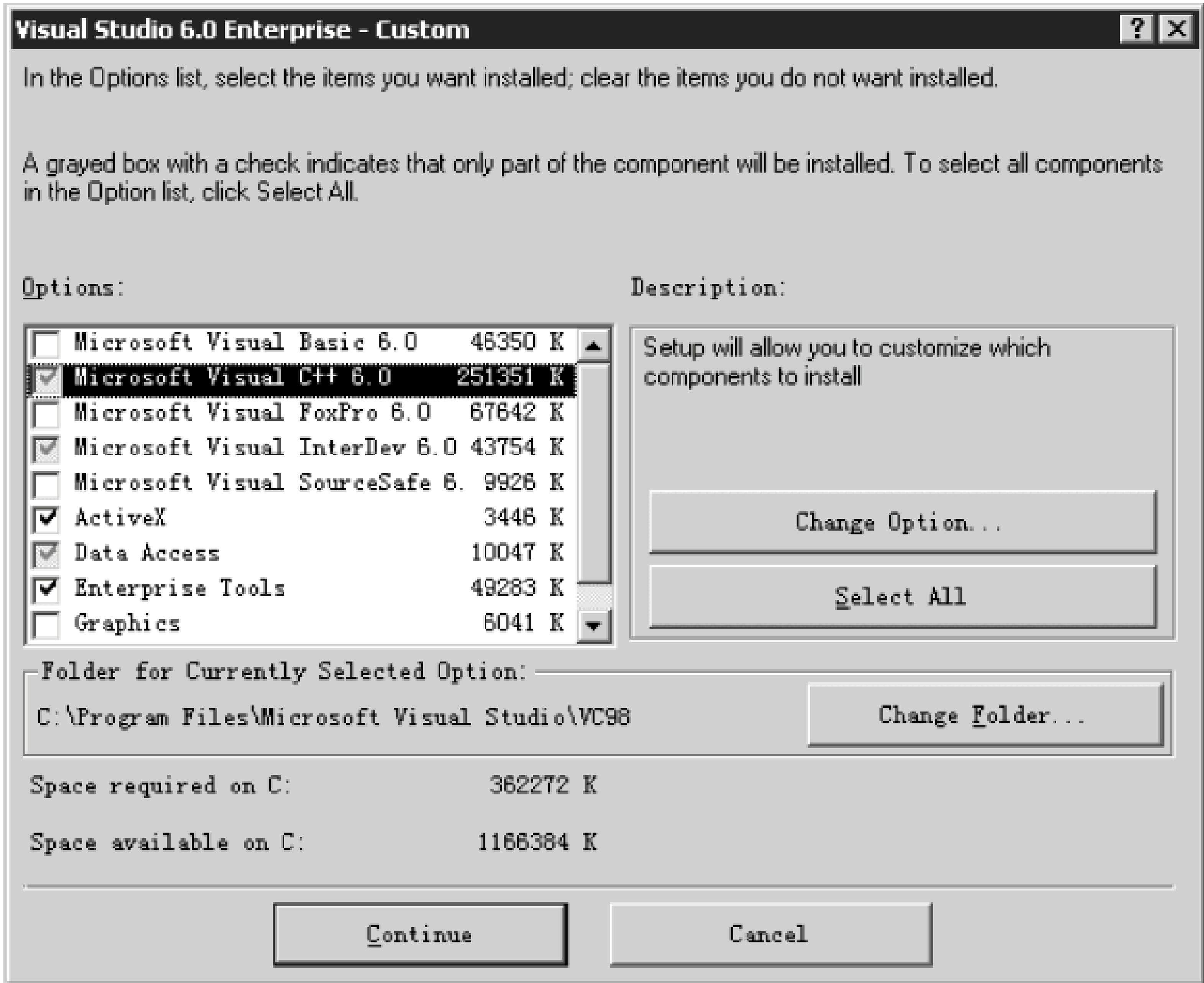


图 2.10　定制开发工具对话框

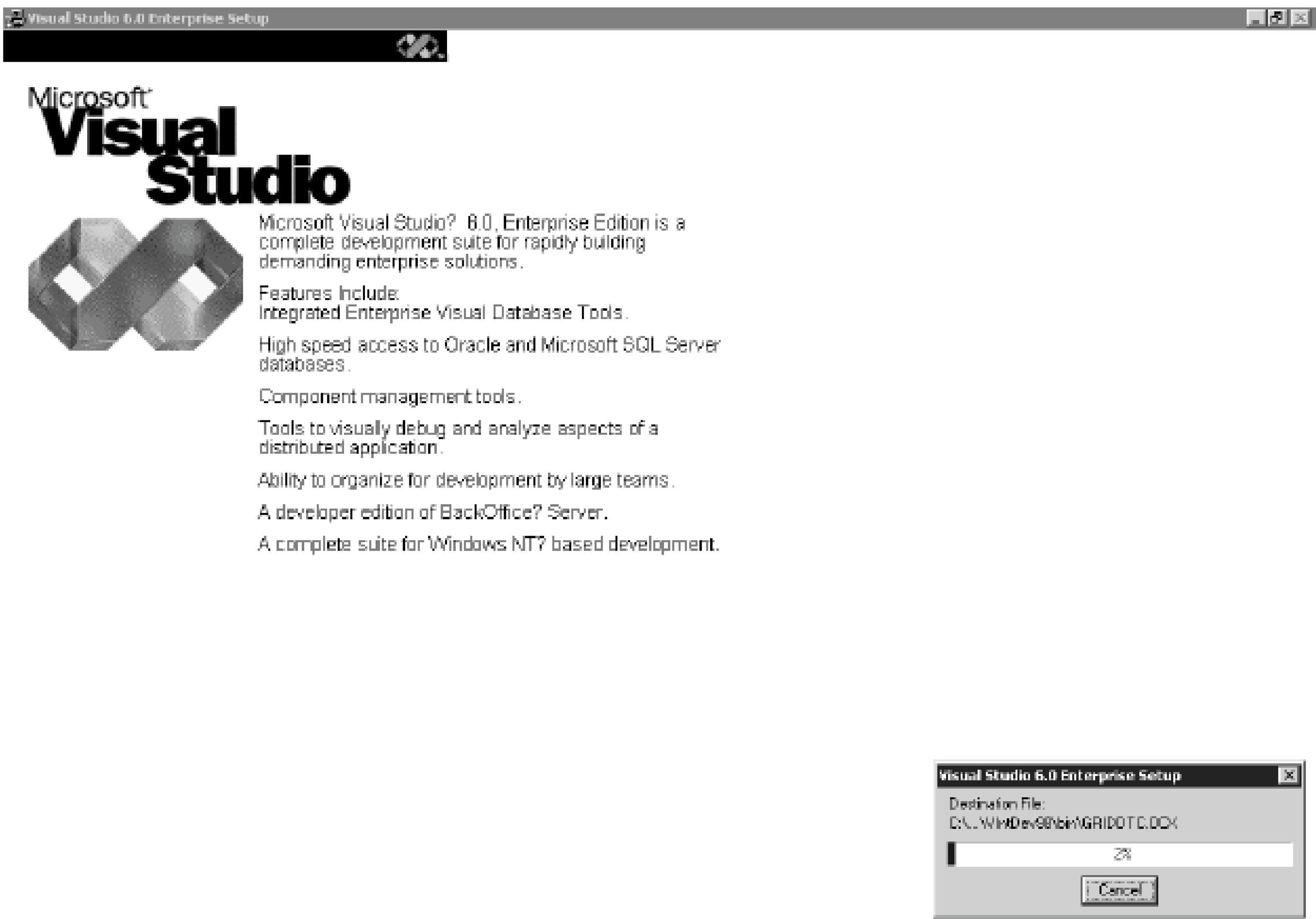


图 2.11　程序安装进行截图

（11）安装完成后，会出现如图 2.12 所示的安装完成对话框。单击 Restart Windows 按钮，重新启动 Windows 操作系统后，Visual C++的安装就完成 80%了。

（12）重新启动系统后，会出现如图 2.13 所示的对话框，这里是要求用户安装 MSDN，即微软的在线帮助系统。其是一个非常实用的帮助系统，用户可以根据自己的需要进行安装，在这里就不再对安装过程进行详细讲解了。单击 Next 按钮进入安装程序的下一步骤。

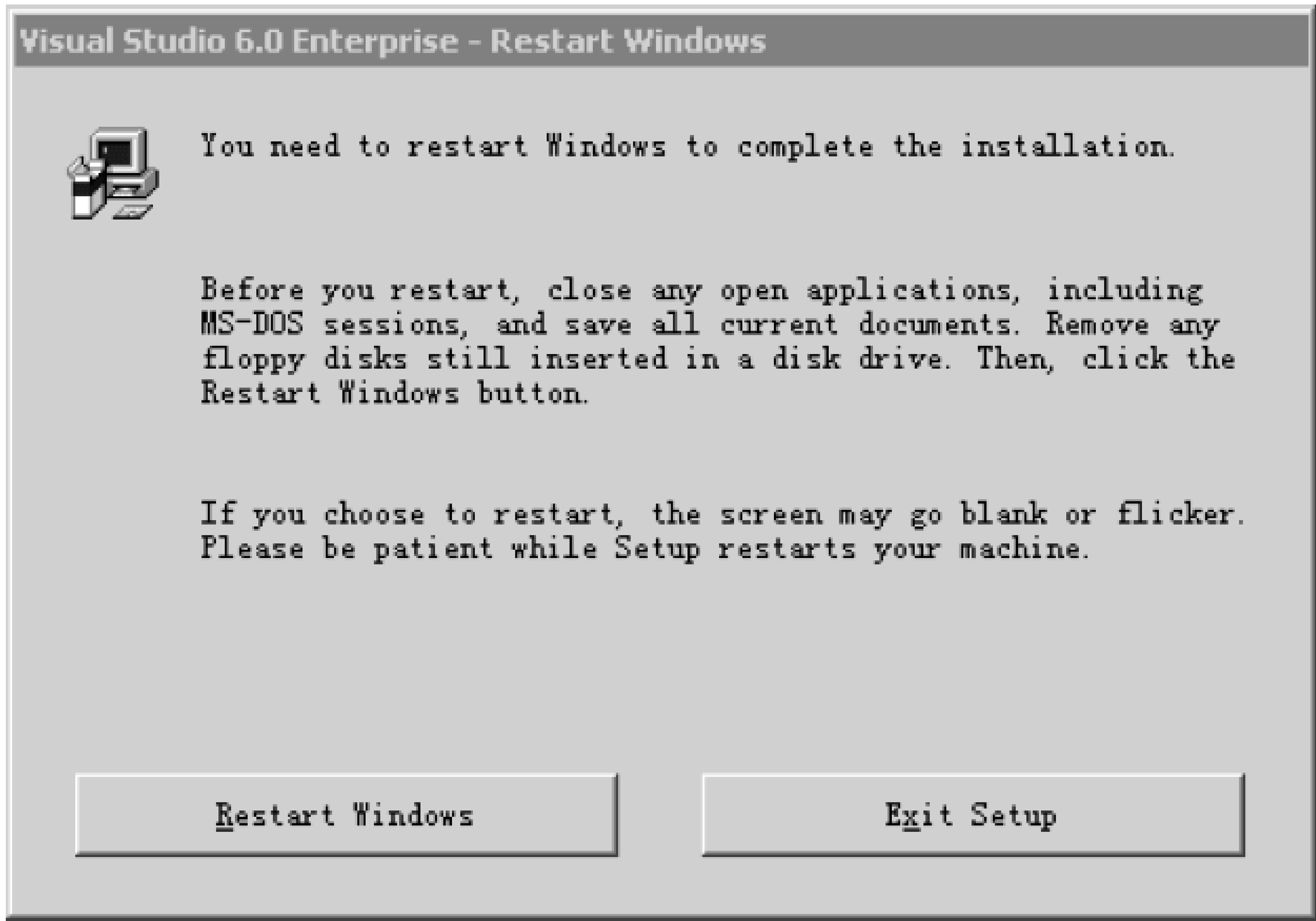


图 2.12　安装完成对话框

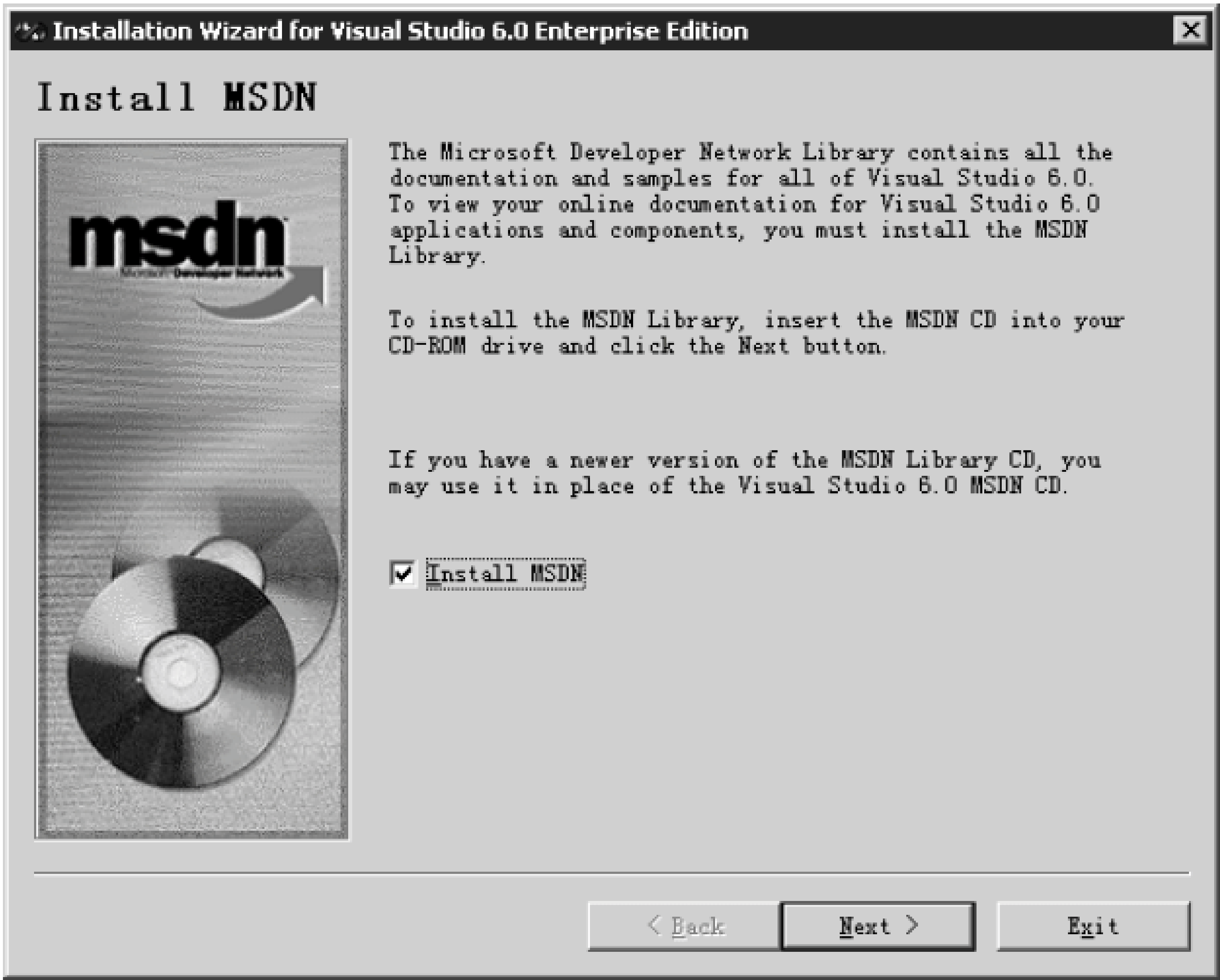


图 2.13　提示安装 MSDN 系统对话框

（13）整个安装步骤完成后，会出现如图 2.14 所示的安装完成注册对话框。直接单击 Finish 按钮，完成 Visual C++ 6.0 的定制安装，不用去注册。

技巧：在安装过程中，一般都采用默认设置，为加快进度，可以直接单击 Next 按钮。

2.2.2　Visual C++的启动

Visual C++安装完毕后，有一个专门的快捷方式作为程序的启动。在默认安装状态下，可以单击桌面的“开始”按钮，将鼠标指针移到“程序”项，然后选择 Microsoft Visual Studio 6.0 | Microsoft Visual C++ 6.0 命令，即可启动 Visual C++ 6.0，如图 2.15 所示。



图 2.14　安装完成注册对话框

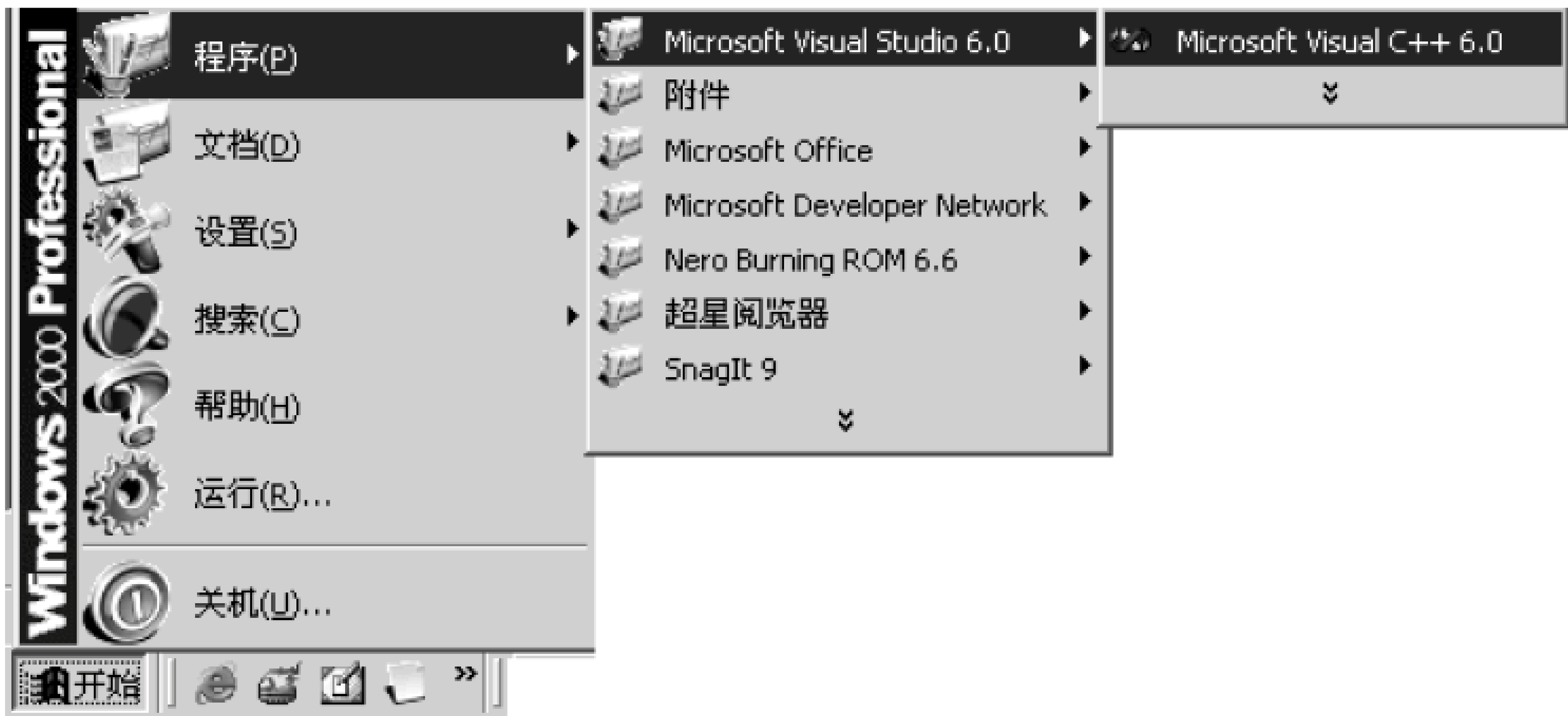


图 2.15　Visual C++ 6.0 的启动菜单项

2.3　部署 Visual C++游戏项目

在一个完整的游戏项目中，文件分为很多种。但因为本书主要讲解使用 Visual C++来开发游戏项目，所以在本节中将只对 Visual C++的各类文件进行说明，其他工具的文件不在这里讲解。而且为了管理方便，在本节中，也将对项目的文件夹命名进行约定。

2.3.1　项目中的各种文件的定义

用 Visual C++工具来开发项目，可以自动生成多种不同类型的文件，文件的类型说明如表 2.1 所示。

表 2.1　自动生成的文件类型表

文 件 类 型	文件后缀名	文 件 说 明
Active Server Page	asp	活动服务器页文件
Binary File	无	二进制文件
Bitmap File	bmp	位图文件
C/C++ Header File	h	C/C++头文件

续表

文 件 类 型	文件后缀名	文 件 说 明
C++ Source File	cpp	C++源文件
Cursor File	cur	光标文件
HTML Page	html\htm	HTML 超文本文件
Icon File	ico	图标文件
Macro File	dsm	宏文件
Resource Script	rc	资源脚本文件
Resource Template	rct	资源模板文件
SQL Script File	sql	SQL 脚本文件
Text File	txt	文本文件
Make file	mak	Make 文件的工程
MFC AppWizard(dll)	dll	MFC 动态链接库
MFC AppWizard(exe)	exe	MFC 应用程序
New Database Wizard	db	SQL 数据库服务器
Utility Project	dsp	空白工程
Win32 Application	exe	Win32 应用程序
Win32 Console Application	exe	Win32 控制台应用程序
Win32 Dynamic-Link Library	dll	Win32 动态链接库
Win32 Static Library	lib	Win32 静态库
Block WorkSpace	dsw	工作组

以上这些都是 Visual C++中常用的一些文件类型，可以自动生成。但还有一些在这个表中没有列出，也是项目开发中常常会用到的文件类型，如表 2.2 所示。

表 2.2　其他文件类型表

文件后缀名	文 件 说 明
ini	配置文件，一般用于配置项目中的各种资源和字符
bin	二进制文件，用于存放资源
dat	数据文件
sys	系统文件，用于 Windows 系统配置
bak	备份文件
obj	源代码编译后自动生成的中间文件

说明：因为在以后的项目实例讲解中也将大量使用到，所以请各位读者一定把这两张表了解清楚，不管是对于以后游戏开发还是其他项目开发都大有裨益。

2.3.2　项目文件夹的定义

因为一个完整项目中涉及的文件较多，常常会把各种不同的文件放在不同的文件夹中，这样整个项目的各种文件管理就比较方便，所以在这里约定本书所讲的各个项目文件夹的名称并按表 2.3 所列名称进行命名。

表 2.3 项目文件夹命名规则

名 称	说 明	名 称	说 明
bin	执行文件和最终生成的各类文件	src	存放各类源文件
lib	存放生成的静态库文件	doc	存放项目相关文件
inc	存放各类头文件	tmp	临时文件存放

2.4 Windows 的窗体

本节将介绍 Windows 窗体的一些基本知识，其中包括什么是窗体及其与应用程序的关系。通过这一节的学习，为后面开发 Windows 游戏做准备。但如果读者已经很熟悉这方面的内容，可以跳过本节，学习后面的内容。

2.4.1 Windows 中的窗体

其实，在微软公司推出 Windows 操作系统时，窗体就已经和系统密不可分了。因为在 Windows 系统中见到的所有东西，例如按钮、图标、菜单、对话框、下拉列表等都是用窗体表示出来的，如图 2.16 所示。

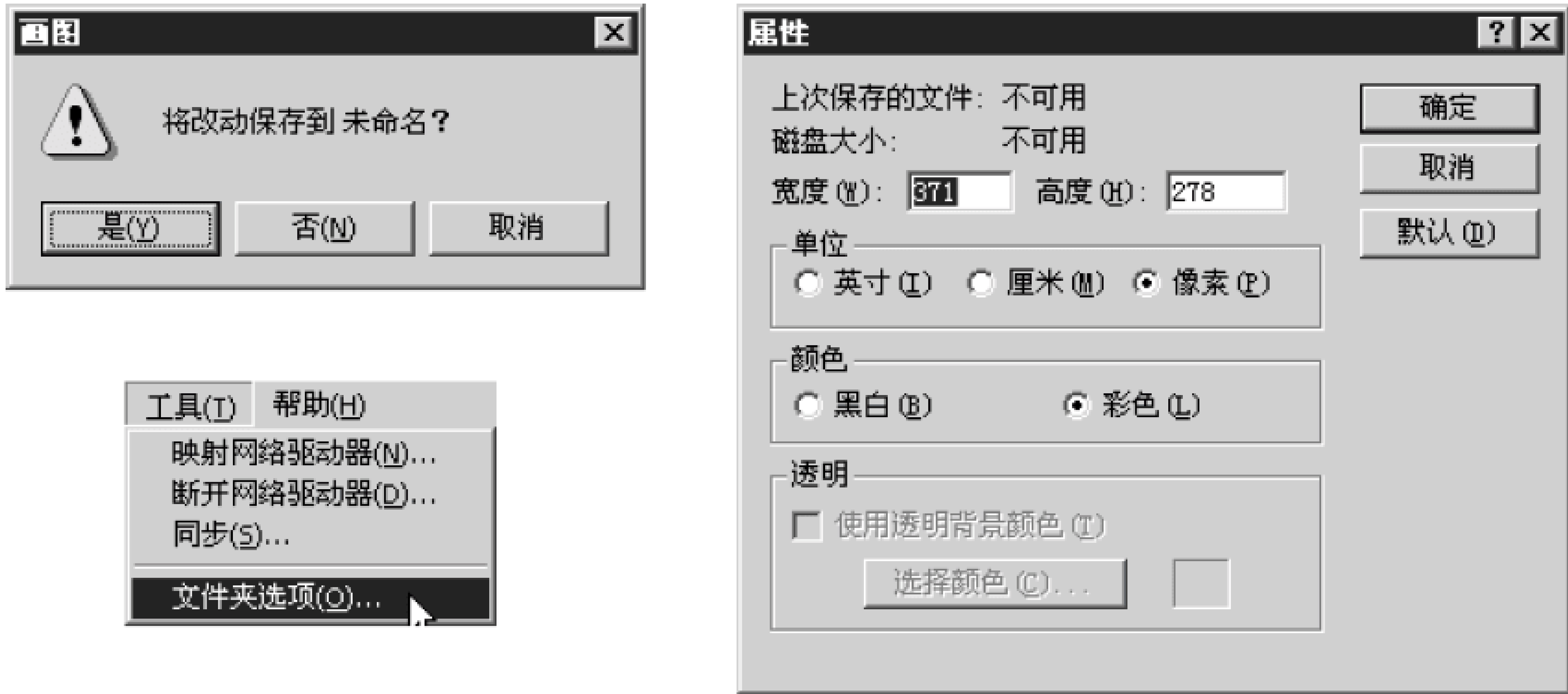


图 2.16 各种各样的系统窗体

窗体是 Windows 操作系统的基础。其是用于生成 Windows 客户端应用程序的框架。可用公共语言运行库支持的任何语言，如 Visual C++、Visual Basic 或者 C 语言等来编写 Windows 窗体应用程序。使用 Windows 窗体的优点如下所述。

- 不必再为界面程序花费大量的时间，而只需要以窗体为基础，并编写相应的代码就可以实现程序的功能，大大提高了编程的效率和灵活性。
- 程序升级总成本较低，对程序进行升级时，只需要将界面与相关的代码进行升级即可符合新的界面和功能需求。
- 窗体支持控件结构，Windows 窗体提供支持控件与控件容器的结构，该结构基于控件和容器类的具体实现。例如，对话框是一个窗体和控件容器，而对话框中的按钮既是窗体，也是控件，如图 2.17 所示。这样的结构显著减少了控件和容器间的交互问题。

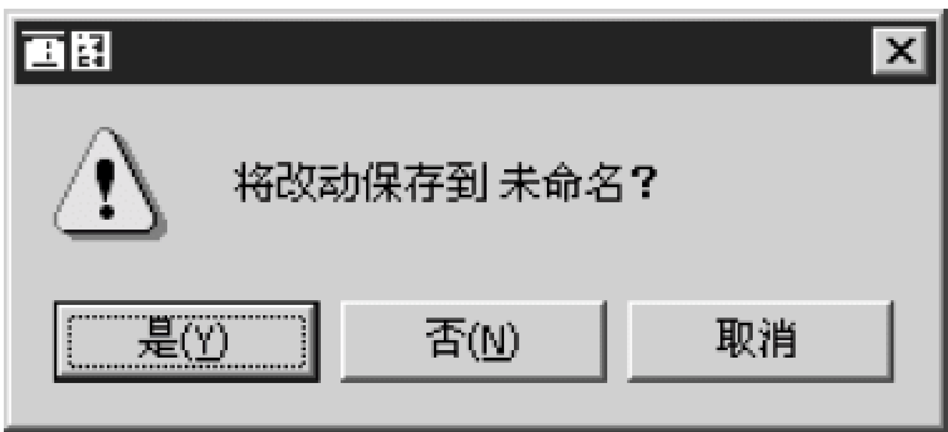


图 2.17 确认提示对话框

- Windows 窗体是与系统内部是紧密结合的，所以其可用于实现系统中所有的内容，从在浏览器中运行的不受信任的控件到安装在用户硬盘上的完全受信任的应用程序，范围十分广泛，具有很高的灵活性能。同时由于其受到系统 API 的限制，所以安全性能也比较高。
- Windows 窗体是 GDI+的载体之一，所以 Windows 窗体包含丰富的图形接口。调用 GDI+中的 API 函数来创建窗体，就可以创建出既统一美观又有特殊效果的用户界面。GDI+是 Windows 图形设备接口，支持 Alpha 混合效果、纹理画笔、高级转换、多格式文本支持等。

2.4.2　应用程序与窗体的关系

Windows 上的应用程序（Application）是指可以在 Windows 系统中运行的程序，是为了完成某项或某几项特定任务而被开发运行于操作系统之上的电脑程序。运行在用户模式，可以和用户进行交互，具有可视的用户界面。

每一个应用程序运行于独立的进程，拥有自己独立的地址空间。利用编程语言能直接调用 Windows 系统的 API 编写的程序，可以在任何装有 Windows 系统的机器上运行，实现在该操作系统中可以编程实现的任何功能。

在 Windows 中，应用程序的扩展名为 exe。因为它是运行在 32 位的操作系统中的，所以地址空间大小为 4GB，也被称为 Win32 应用程序。

读者一定要注意不要把应用程序与应用软件的概念混淆。应用软件是指程序与其相关文档或其他资源的整体集合，应用程序只是应用软件其中的一个组成部分。

例如，一个游戏软件包括应用程序（exe）和其他图片（jpg、bmp）、音效（MP3、wav）等资源，那么这个程序（exe）被称为“应用程序”，与其他文件（图片、音效等）一起被合称为“游戏软件”。

Win32 应用程序包含 Windows 窗体程序和控制台程序两种。我们这里所讲的窗体都是指出现在 Windows 窗体程序中的窗体或者控件。应用程序的功能是由内部的函数来实现的，而界面上的输入与输出（即与用户的交互），一般都是依靠窗体表现出来的。如对话框、提示框、按钮、菜单、文本框等。

注意：*在 Windows 操作系统中，所有的控件和对话框都是归属于窗口类 CWnd（将在后面的内容中讲到），这点一定要记住。*

2.5　使用 Visual C++开发工具

在前面的章节中，我们学习了如何安装 Visual C++开发工具。本节将介绍如何使用 Visual C++开发工具来创建一个 Windows 窗口应用程序。

2.5.1　Visual C++开发工具的主界面

启动 Visual C++后，就可以看到如图 2.18 所示的主窗口。其中包括了标题栏、菜单栏、

工具栏、工作区窗口、代码编辑区窗口、输出窗口、状态栏。其中标题栏比较特殊，位于主窗口的上方，显示当前程序的标题名称。

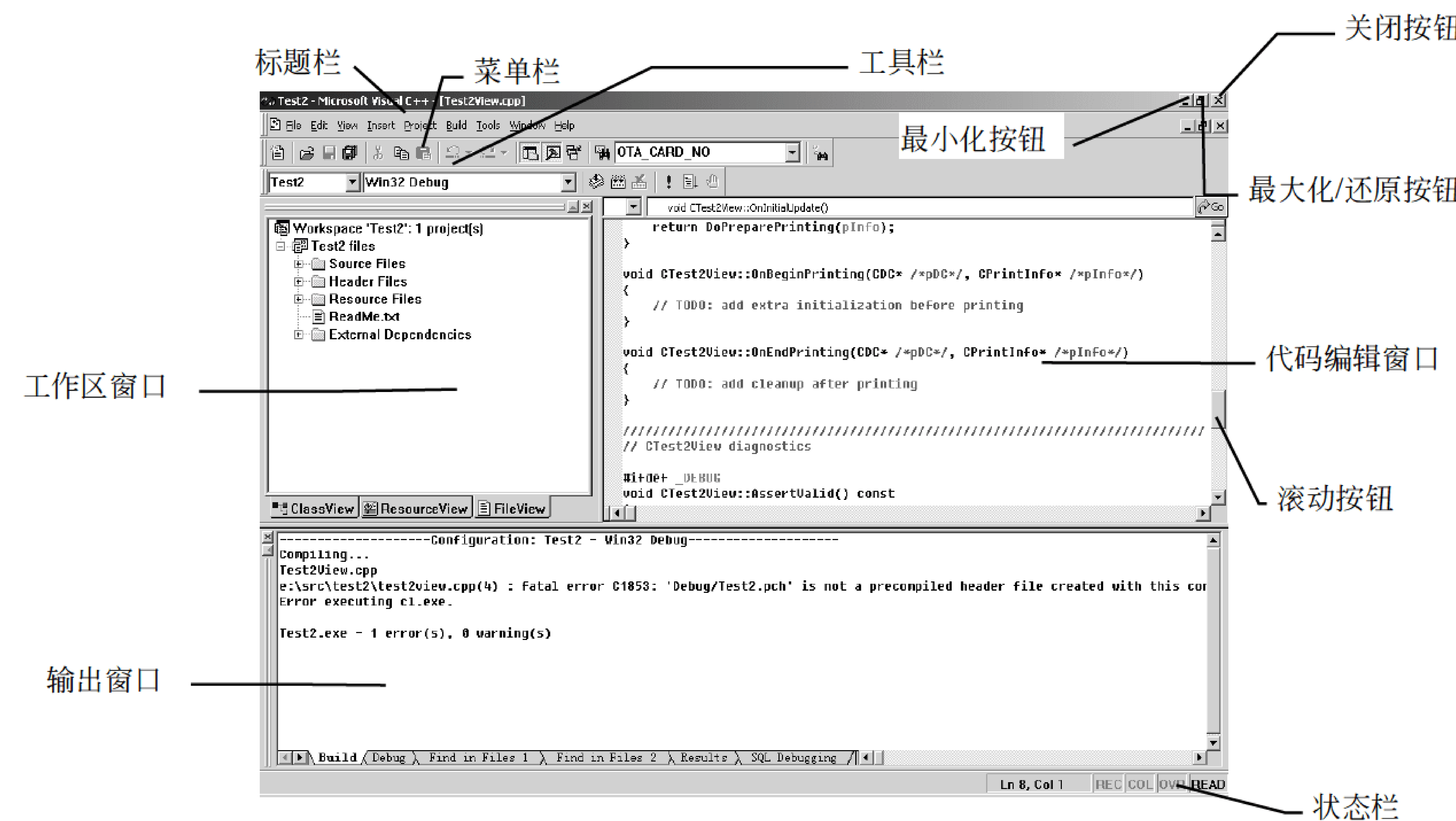


图 2.18　Visual C++开发工具的主窗口

注意：熟悉 Visual C++开发工具的操作界面，有利于以后在使用该工具时提高工作效率。在阅读相关说明时，也便于理解。

2.5.2　使用向导创建项目

现在就可以使用“创建向导”来创建一个空的 Windows 窗体应用程序项目。启动 Visual C++向导的方法如下：

（1）启动 Visual C++集成开发环境。

（2）选择 File | New 命令，打开 New 对话框。

（3）选择 Projects 标签，进入 Projects 选项卡。在其中选择 Win32 Application 选项。然后在 Project name 中输入一个名字作为整个项目的名称。

（4）把保存路径 Location 设置为有效路径 e:\src\Demo，如图 2.19 所示。

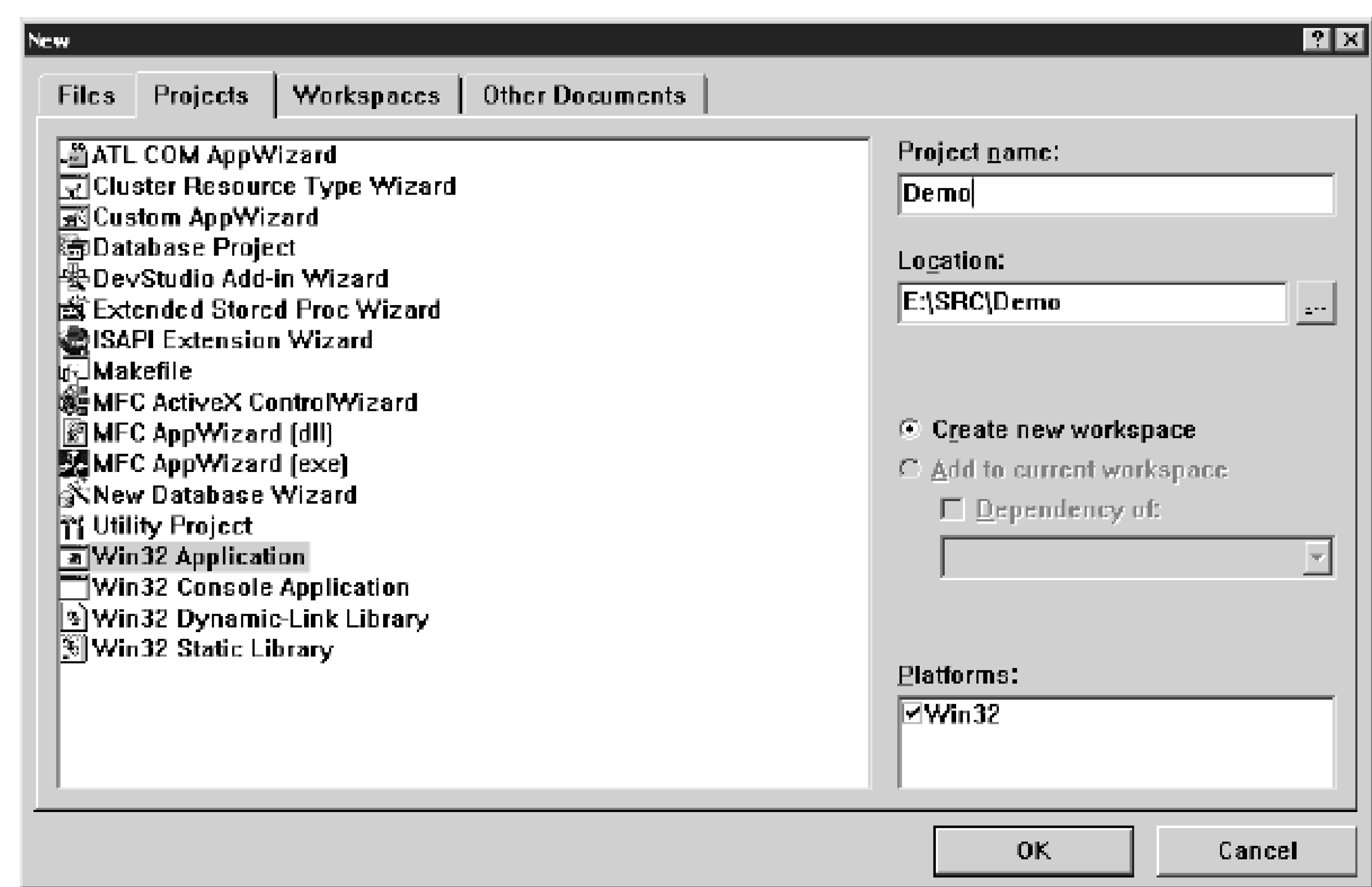


图 2.19　New 对话框

（5）单击 OK 按钮，即可启动“Win32 窗口应用程序项目创建向导”。

通过 Visual C++工具的向导，不仅可以免去了很多复杂的项目设置过程，同时还可以提高项目创建的效率，方便了项目的管理。在 2.5.3 节中，我们将学习到如何创建一个真正的 Win32 窗口应用程序项目。

2.5.3　创建一个 Hello World 程序

现在我们需要创建一个能显示 HelloWorld 的窗口应用程序项目，创建项目的方法如下所述。

（1）在成功启动了向导后，选中向导选项的 An empty project 项。表示当前创建的是一个不含任何文件的空窗口应用程序项目，如图 2.20 所示。

（2）单击 Finish 按钮，Visual C++就开始 Win32 窗口应用程序项目的创建工作。

（3）创建完成后会得到一个项目创建报告，如图 2.21 所示。单击 OK 按钮，关闭报告对话框。

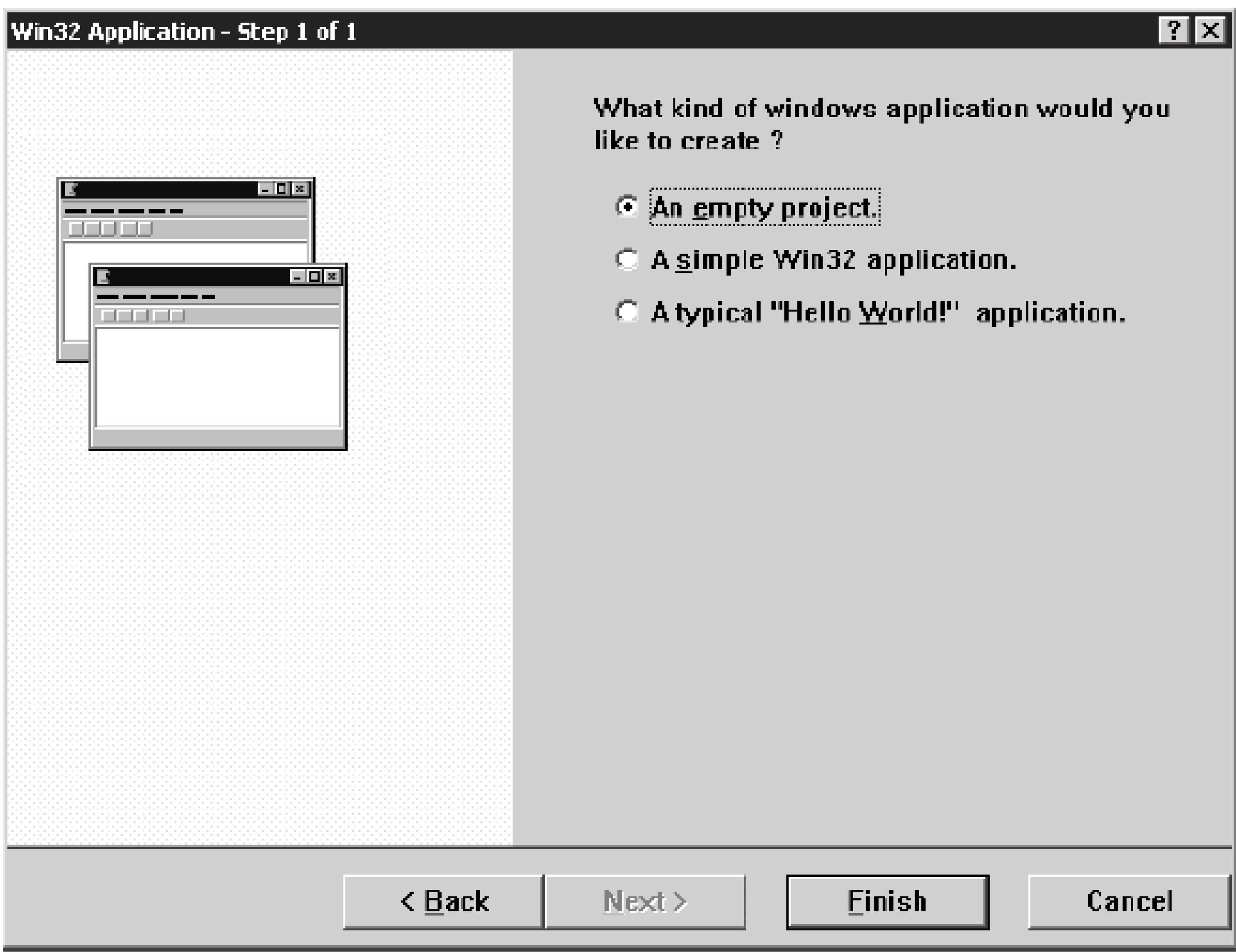


图 2.20　Win32 窗口应用程序项目创建向导

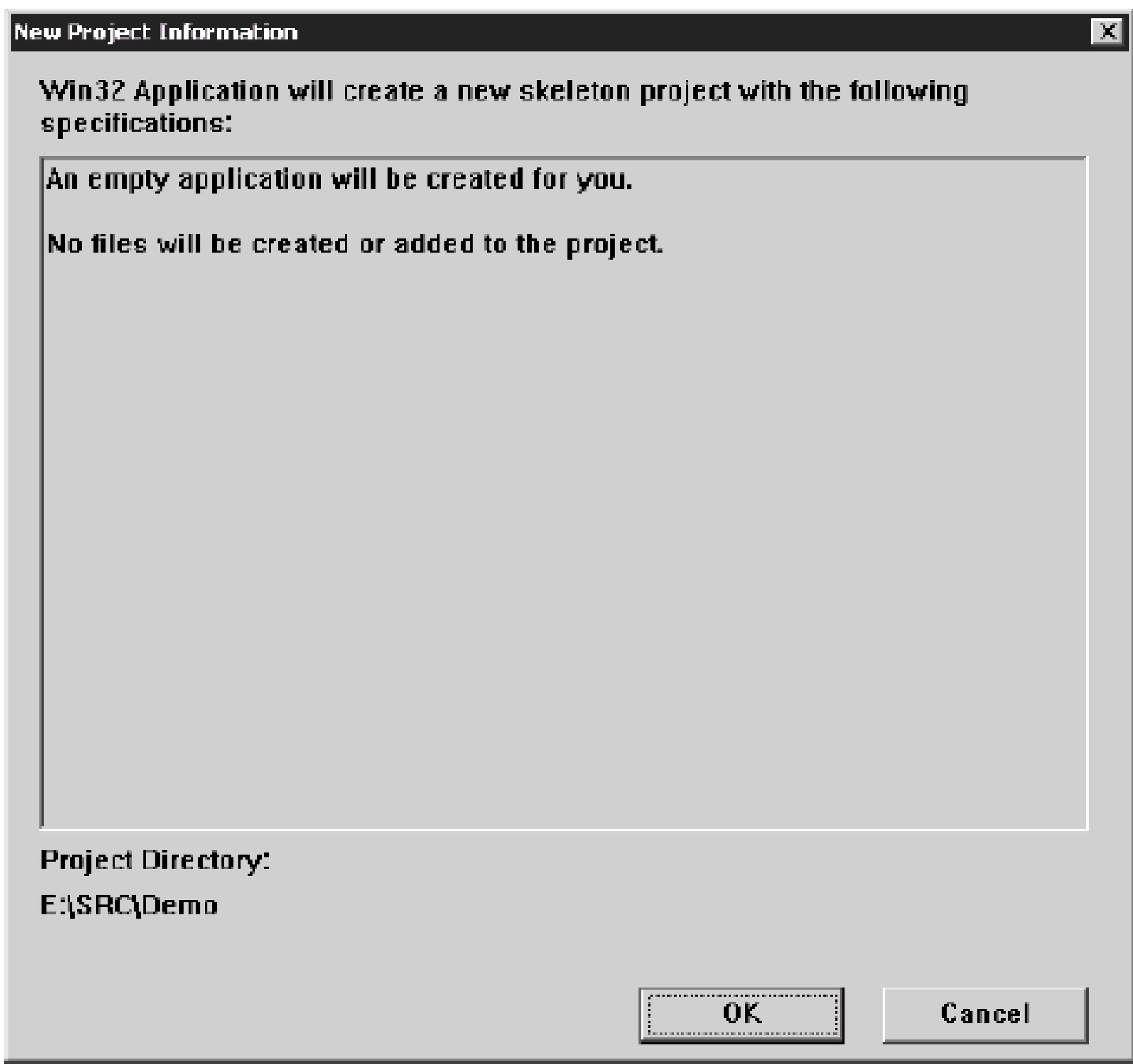


图 2.21　项目创建报告

通过前面的方法，可以得到一个只包含应用程序框架的项目，其不包含任何源代码。没有源代码的程序什么都做不了，就好像没有安装系统的电脑一样。要达到前面的要求，所以必须给当前工程添加一段源代码。

在 Visual C++中，通常的源代码保存在以 cpp 为后缀的文件中，这种文件也被称为源文件。现在就来给当前项目添加一个源文件，并输入相关的源代码，让其能够完成前面的要求。添加源文件的方法如下所示。

（1）选择 File | New 命令，在 New 对话框中选择 Files 标签，进入 Files 选项卡。

（2）选中 C++ Source File 项，在 File 文本框中输入 Demo 作为 C++源文件的名称，并选中 Add to project 复选框（即添加到当前 Demo 项目中），如图 2.22 所示。

技巧： 要把源文件的路径设置为当前目录下的 src 文件夹中，方便项目文件的管理。如果目录没有这个文件夹，创建时会出错，所以一定要先在项目目录中把这个文件夹建好。

（3）在代码编辑窗口中输入代码，内容如代码 2.1 所示【代码参考：光盘的源代码\C02\Demo.dsp】。在这里读者不需要理解这段代码的详细意思。相关内容将在后面的讲解

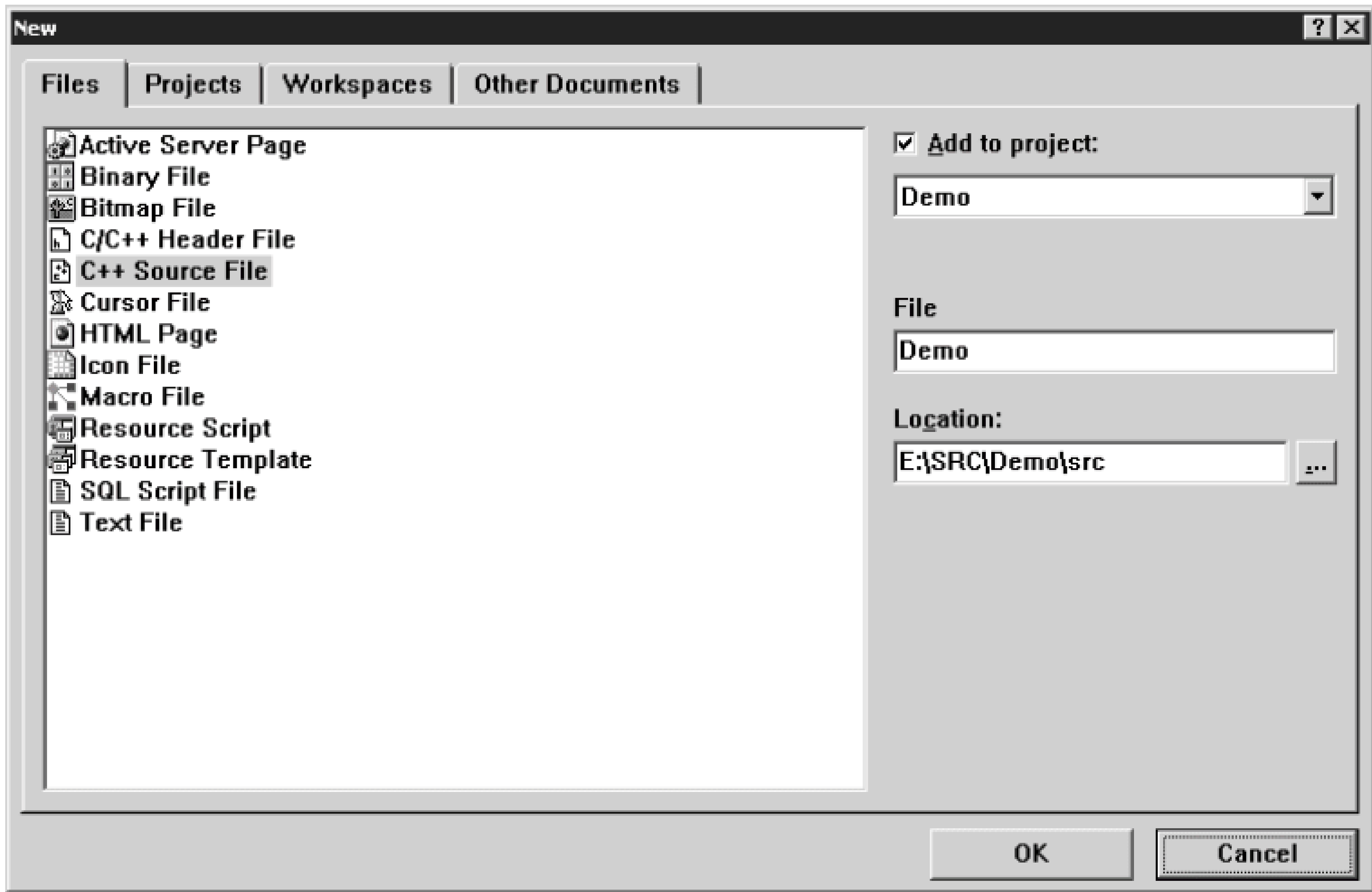


图 2.22　添加源文件到当前项目

中说明。

代码 2.1　HelloWorld 代码

```
01  #include <windows.h>                //一个 Windows 应用程序应该包含的头文件
02  #include <stdio.h>                  //标准输入输出流文件
03  LRESULT CALLBACK WinSunProc         /*声明一个回调函数*/
04  (
05   HWND hwnd,                         /*窗口的句柄*/
06   UINT uMsg,                         /*窗口的消息*/
07   WPARAM wParam,
08   LPARAM lParam
09  );
/************************************************************************
 WinMain:Windows 程序的入口函数
创建一个完整的窗口需要经过下面四个操作步骤：设计一个窗口类;注册窗口类;创建窗口;显示及
更新窗口
************************************************************************/
10  int WINAPI WinMain
11  (
12   HINSTANCE hInstance,               //实例句柄，当前应用程序的实例句柄
13   HINSTANCE hPrevInstance,           //默认这个参数为 NULL
14  LPSTR lpCmdLine,                    //储存一个命令行参数
15  int nCmdShow)
16  {
17      WNDCLASS wndcls;                //定义一个窗口对象
18      wndcls.cbClsExtra=0;            //指定额外内存空间
19      wndcls.cbWndExtra=0;            //指定额外内存空间
        //指定窗口背景色
20      wndcls.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
        //设置光标样式
21      wndcls.hCursor=LoadCursor(NULL,IDC_CROSS);
        //设置图标样式
22      wndcls.hIcon=LoadIcon(NULL,IDI_ERROR);
23      wndcls.hInstance=hInstance;     //指定窗口实例句柄
24      wndcls.lpfnWndProc=WinSunProc;  //指定窗口函数，即窗口主处理函数
        //窗口类名称
25      wndcls.lpszClassName="Visual C++ Game";
26      wndcls.lpszMenuName=NULL;       //菜单
```

```
27      wndcls.style= CS_HREDRAW|CS_VREDRAW;
28      RegisterClass(&wndcls);             //注册窗口类
29      HWND hwnd;                          //声明窗口句柄
30      hwnd=CreateWindow                   /*创建窗口，但这里的窗口是不会显示的*/
31      (
32          "Visual C++ Game",              /*已注册窗口类的名称*/
33          "Visual C++ 游戏开发",           /*窗口标题*/
34          WS_OVERLAPPEDWINDOW,            /*窗口风格*/
35          200,                            /*窗口位置的横坐标*/
36          200,                            /*窗口位置的纵坐标*/
37          600,                            /*窗口的宽度*/
38          400,                            /*窗口的高度*/
39          NULL,
40          NULL,
41          hInstance,                      //实例句柄
42          NULL);
        //在这里真正显示窗口
43      ShowWindow(hwnd,SW_SHOWNORMAL);
44      UpdateWindow(hwnd);                 //更新显示
/*************************************************************************/
/* 初始化工作完成后，WinMain 进入所谓的消息循环*
/*************************************************************************/
45      MSG msg;
46      while(GetMessage(&msg,NULL,0,0))
47      {
48          TranslateMessage(&msg);         //转换键盘消息
49          DispatchMessage(&msg);          //分派消息
50      }
51      return 0;
52  }
/*************************************************************************/
/*窗口函数。窗口函数通常利用 switch/case 方式判断消息的种类，
以决定处置方式，由于其是被 Windows 系统所调用的，所以这是一种 call back 函数*/
 /************************************************************************/
53  LRESULT CALLBACK WinSunProc(
54       HWND hwnd,                         /*窗口句柄*/
55       UINT uMsg,                         /*消息*/
56       WPARAM wParam,                     /*参数 1*/
57       LPARAM lParam                      /*参数 2*/
58       )
59  {
60       switch(uMsg)                       /*判断消息类型*/
61       {
62       case WM_PAINT:                     /*更新窗口消息*/
63           HDC hDC;                       /*定义 DC 设备*/
64           PAINTSTRUCT ps;
65           hDC=BeginPaint(hwnd,&ps);      /*得到设备 hDC*/
66           TextOut(hDC,200,0,"Visual C++ 游戏开发",strlen("Visual C++ 游
             戏开发"));
67           EndPaint(hwnd,&ps);
68           break;
69       case WM_CLOSE:                     /*当单击关闭按钮时，产生关闭消息*/
70           if(IDYES==MessageBox(hwnd,"是否真的结束？","游戏开发",
             MB_YESNO))
71           {
72               DestroyWindow(hwnd);       /*单击“确定”按钮，销毁窗口*/
```

```
73            }
74            break;
75        case WM_DESTROY:                /*销毁窗口消息*/
76            PostQuitMessage(0);         /*退出程序*/
77            break;
78        default:
    //在 default:处必须调用 DefWindowProc,这是 Windows 内部默认的消息处理函数
79            return DefWindowProc(hwnd,uMsg,wParam,lParam);
80            }
81        return 0;
82    }
```

读者是不是被这么多的代码吓了一跳？其实不用担心，这些代码都是模块化的，后面将会讲解到，在这里读者大可不必去详细了解。只需要知道代码的第 66 行，就是调用 TextOut()函数实现输出需要的 HelloWorld 字符串。

技巧：Windows 中的每个窗口都有自己的消息循环机制。

读者可以按键盘上的 F5 键或者单击编译工具栏中的运行按钮 来运行程序，最后程序执行效果如图 2.23 所示。



图 2.23　程序运行效果图

2.5.4　工程文件的配置

通过 2.5.3 节中的示例程序，可以输出一段文字到屏幕的窗口上。虽然已经完成了 Win32 窗口应用程序的开发，但是项目工程生成的各种文件，并没有按照前面 2.3 节的“各种不同类型的文件放在不同的项目文件夹中”的要求。所以本节就是指导读者如何把各种不同的文件放入对应的文件夹中。

在这里只是简单地把执行文件放置的位置进行管理，其他文件类型将在后面实例演示中一块讲解。配置应用程序执行文件的输出文件夹和工作路径方法如下所述。

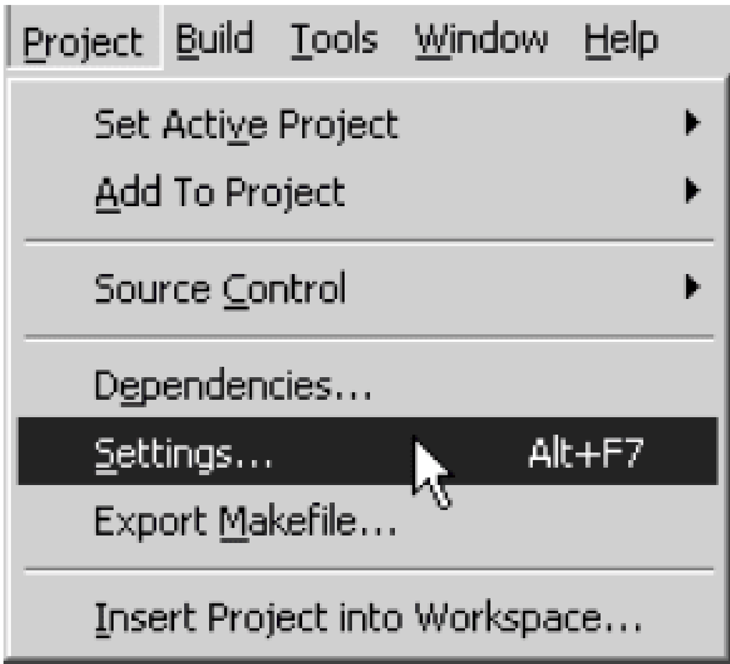


图 2.24　选中 Setting 菜单项

（1）选中主界面中的 Project | Settings 命令，如图 2.24 所示。

（2）在弹出的“项目设置”对话框中，选择 Link 标签，进入 Link 选项卡，如图 2.25

所示。把 Output file name 文本框中的路径修改为 bin/xxxx.exe，这里的 xxxx.exe 是指最后执行文件的名称。

（3）选择 Debug 标签，进入 Debug 选项卡，如图 2.26 所示。把 Working directory 文本框中的路径也指定为“.\bin”目录，这样就完成了把执行文件自动输出到 bin 文件夹中的要求。

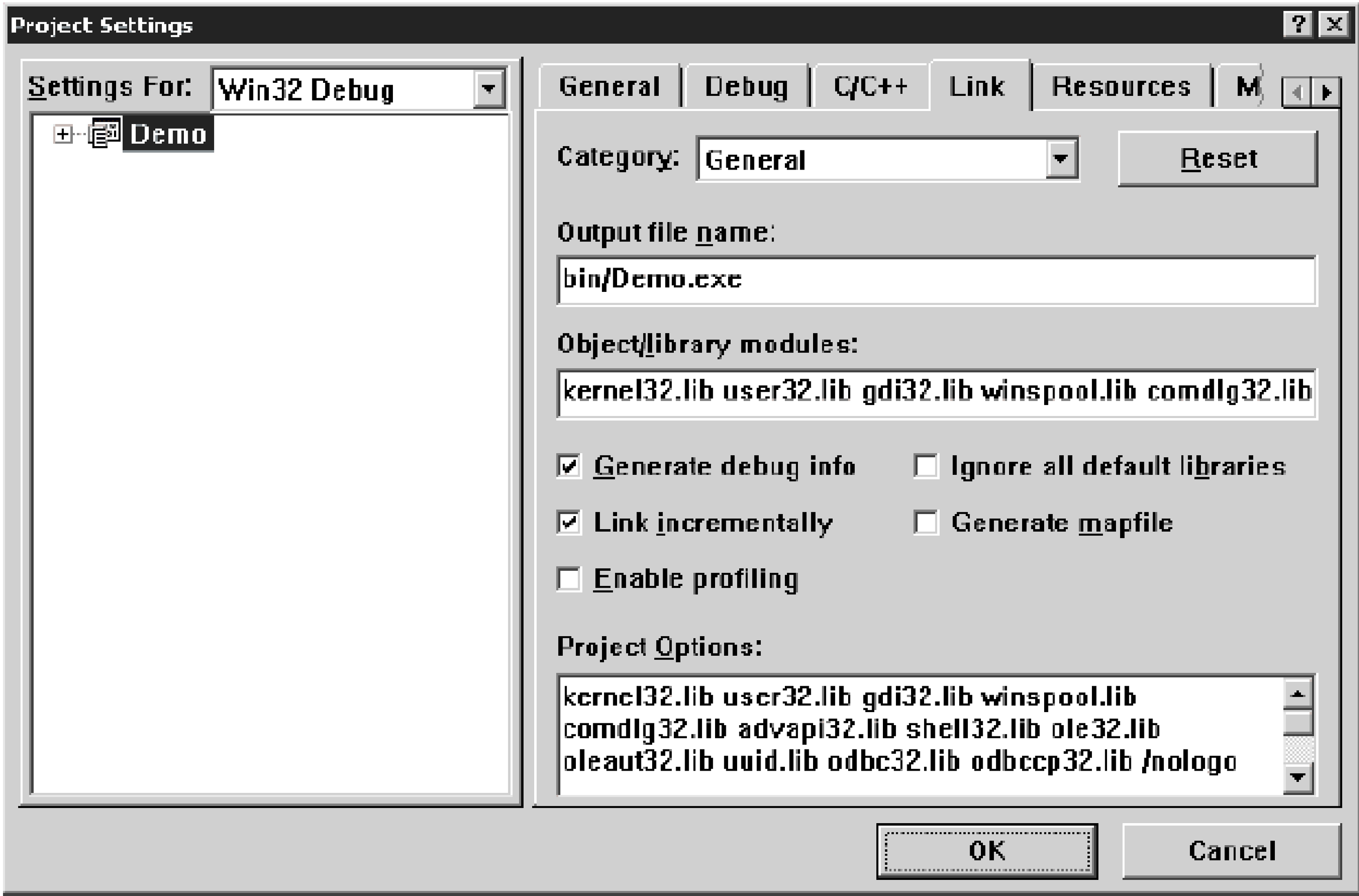


图 2.25　项目设置对话框的 Link 标签

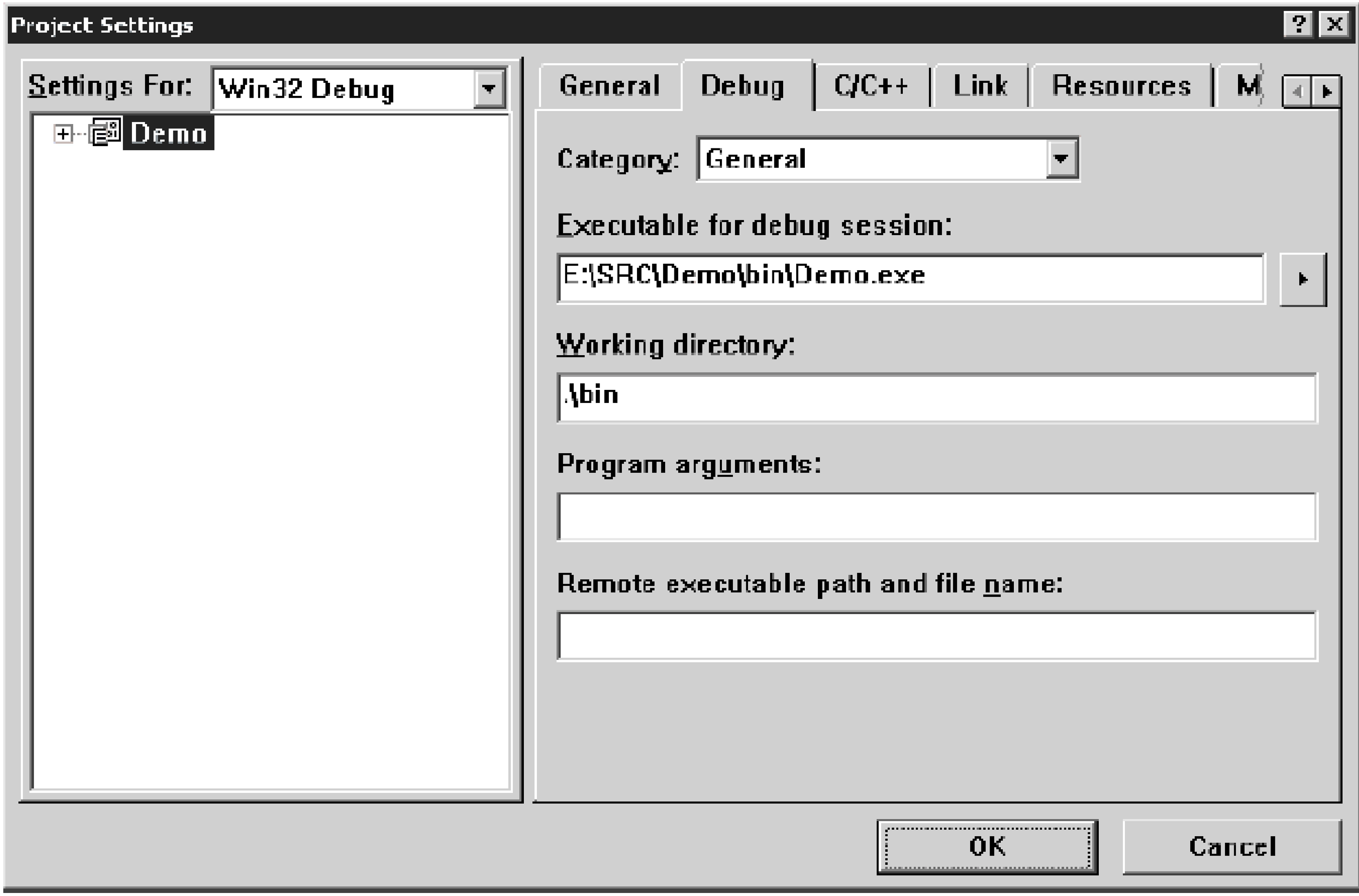


图 2.26　项目设置对话框的 Debug 标签

2.6　总　　结

通过本章内容的学习，相信读者对使用 Visual C++工具开发游戏的兴趣将会大大地提高。本章简单地讲述了 Visual C++开发工具的特点和历史，并介绍了如何安装与启动 Visual C++。同时对于项目中各种文件与文件夹的安排也进行了简单的说明。这些都是实际项目应用中的经验，可以帮助读者快速地提高开发水平。

不仅如此，在本章中还详细讲解了 Windows 的窗体开发的背景知识部分，并给出一个简单的示例程序。这就意味着你将要开始学习一些真正的 Windows 窗体游戏设计的知识了。在第 3 章中，读者将了解到：

- C++编程语言的特点及历史。
- C++的各种字符与数据类型。
- C++中的运算符与表达式。
- 常量、变量及控制语句。
- C++的指针和数组。
- C++的类与成员。
- 神奇的运行符重载机制。
- 常用的 C++编程规范。

现在打起精神来，因为下面是真正动手学习 C++基础知识的时间了，而且在这个过程中事情也会变得更加复杂起来。

第 3 章　C++编程语言基础

如同在第 2 章中所提到的那样，Visual C++只是一个开发工具，并不能实现软件应具备的所有功能。因此，了解 C++编程语言的各种特性以及基础语法，对于游戏编程高手而言是非常必要的。

因此，本章将主要帮助读者熟悉 C++编程语言的优点及基本语法。把这些基本语法与 Visual C++开发工具相结合就是巩固所学知识的最好方法。但愿本章所列举的材料能够很好地帮助读者理解前一章中所学习的内容，同时能够建立起一定的 C++编程语言基础。如果您已经是一位 C++编程人员，那么请跳过本章，继续学习后面的内容。如果只是一个初学者，那么请一定要详细阅读这一章，并对每一节的示例代码进行实践。只有这样才能够真正的走进 C++程序设计的大门。

其实，有关 C++编程语言的内容在前面已经有所涉及，如第 2 章中的 HelloWorld 程序就演示了编程语言是如何实现文字输出的。本章将通过一些更加详细的内容来帮助读者了解 C++编程语言。为了保证这些应用程序代码能够正常运行，可能在前面的代码简单的涉及后面代码中的内容，读者只需要明白代码中本节所包含的内容即可。在阅读完本章的全部内容后，再回过头来看就会明白。

本章主要涉及的内容如下：

- C++编程语言的优点及其发展过程：了解 C++编程语言的优点及其发展。
- C++中的各种字符：掌握什么是 C++中的关键字、标识符等概念。
- 常用数据类型：明白常用数据类型的范围及其定义。
- 运算符与表达式：知道什么是表达式、运算符，及其如何使用。
- 常量与变量：掌握变量与常量的定义及区别。
- 控制语句：掌握基本的语法，并熟悉相关的控制语句。
- 数组与指针：知道如何使用数组及指针。
- C++类的特性：了解 C++中类的特性及其主要的成员。
- 运算符的重载：了解其基本方法，明白其优点。
- 编程规范：掌握基本的编程规范，对自己编程或者阅读别人的代码大有裨益。

3.1　C++编程语言是什么

在学习 C++这门编程语言之前，确实有必要了解一下这种编程语言的发展过程、优点及特性，这样才能对一种编程语言有一个全面的了解。

3.1.1　C++语言的由来

C++语言起源于 C 语言。在 1973～1979 年间，C 语言迅速成为应用最广泛的系统程序设计语言。然而，由于 C 语言也存在一些缺陷，例如类型检查机制相对较弱、缺少支持代码重用的语言结构等，造成用 C 语言开发大程序比较困难。为了克服 C 语言存在的缺点，在 1980 年，由美国贝尔实验室在 C 语言的基础上，开始对 C 语言进行改进和扩充，并将“类”的概念引入了 C 语言，构成了最早的 C++语言（1983 年）。

后来 C++中又引进了运算符重载、引用、虚函数等许多特性，并使之更加精炼。由贝尔实验室开发出的这种过程性与对象性相结合的程序设计语言，直到 1983 年正式取名为 C++。以后又经过不断的完善和发展，由美国国家标准化协会 ANSI 和国际标准化组织 ISO 一起进行了标准化工作，并于 1998 年正式发布了 C++语言的国际标准（ISO/IEC:98-14882）成为目前的 C++语言。

简单地说，C++语言是在 C 语言的基础上引入了面向对象的机制而形成的一门计算机编程语言。C++继承了 C 语言的大部分特点：一方面，C++语言将 C 语言作为其子集，使其能与 C 语言相兼容；另一方面，C++语言支持面向对象的程序设计，如类的概念和性质。这就是对 C 语言的重要改进。

3.1.2　C++语言的特点

C++语言的特点大致有如下 3 点：

- C++语言是一种面向对象的程序设计语言。其模仿了人们建立现实世界模型的方法。C++语言的基础是对象和类。现实世界中客观存在的事物都被称为对象。例如，一辆汽车、一家百货商场等。C++中的一个对象就是描述客观事物的一个实体，其是构成信息系统的基本单位。类（class）是对一组性质相同对象的描述，是用户定义的一种新的数据类型，也是 C++语言程序设计的核心。
- C++是 C 语言的超集。其不仅包含了 C 语言的大部分特性，例如指针、数组、函数、语法等。其还包含面向对象的特点，例如封装、继承、多态等。
- C++是程序员和软件开发者在实践中创造的。

注意：由于 C++语言来源于 C 语言，所以不管哪种 C++结构都可以用 C 语言实现。但 C 语言的结构，C++不一定可以实现。

3.2　C++中的各种字符

世界上所有的语言都有自己的字符表示和组合，像平时常常接触到的中文和英文都是如此。所以为了让大家都能够使用 C++编程语言，其设计者也给这种语言定义了自己的字符表示和组合。从本节起，才真正开始介绍 C++编程语言的基础内容。

3.2.1　标识符与关键字

在 C++中，有一套用来表示程序中的变量、常量、数据类型及语法关键字的符号，这些符号被统称为标识符。

标识符的命名有如下几点规则需要遵循：

- 标识符的第一个字符必须是字母或者下划线。
- 标识符中不应有除字母、数字和下划线以外的字符。
- 标识符的长度一般不超过 31 个字符。
- C++中的标识符可以大写，也可以小写。不过大写和小写是有区别的，即对应的大小写字母会被当作不同的标识符。例如 Good 和 good 是被当作不同的标识符，Hello 和 HELLO 也是不同的标识符。

例如：下面这些就是合法的标识符的例子。

```
Inno  Result  Good  At_This  GG_88  nCount
```

而如下这些就是不合法的标识符例子。

```
2SD  A!bc  GG*88  good-bye
```

在 C++的标识符中，有些单词组合是不能由用户声明的，其是由 C++编程语言本身保留使用，具有特殊的含义。一般用于表示固定语句、预定义类型说明、预定义函数等。这种标识符被统称为关键字或者保留字。

有了这些关键字，C++编译器才能正确识别输入的程序代码是如何分隔的，这就好像写应用文时为了突出重点，常常把关键字或者词进行标注一样。表 3.1 列出了一些 C++中常用的关键字（只是一部分），请读者注意。

表 3.1　常用关键字

public	private	class	delete
new	const	void	unsigned
int	long	short	char
true	false	float	double
main	try	this	Enum
bool	sizeof	operator	Struct
_asm	_int8	_int16	_int32

注意：在 Visual C++中，为了让用户阅读方便，所有的保留关键字都会被自动高亮标识出来。读者可以在 Visual C++的源文件中输入一些关键字试一试。

3.2.2　分隔符与注释符

C++中除了 3.2.1 节所说的标识符外，还有两种起特殊作用的符号。一种是用来分隔代码语句的，被称为分隔符；另一种是起说明作用的，被称为注释符。

1．划分语句的分隔符

其中分隔符又被称为 C++中的标点符号。用来将单词或者程序分隔，其表示某个程序的结束和另一个程序的开始。C++中包括如下几种分隔符。

- 空格符：用来作为单词与单词之间的分隔符。
- 逗号：用来作为说明多个变量的分隔符，或者多个参数（将在后面的章节讲解）之间的分隔符。
- 分号：用来作为 C++中语句的结束分隔符。
- 花括号：用来构造程序实体的分隔。

2．使语句无效的注释符

注释在程序代码中起到对程序语句注解和说明的作用。其目的是为了代码设计者或审查者方便阅读程序代码。对计算机而言其是无效的，在程序编译时，注释会被自动从程序代码中忽略掉。换句话说，这就好比大家在读书时所做的读书笔记，可以标记在书上，也可以标记在笔记本上，但对书的内容（代码）并没有任何影响，只是起一个辅助说明的作用。

在 C++中采用如下两种注释方法。

（1）使用/*和*/括起来进行注释，在/*和*/之间的字符都被作为注释符处理，适用于多行注释信息，如图 3.1 所示。图 3.1 中从/*开始一直到*/结束的两行字符都是被当作注释信息处理的。

```
void CList::disp()
{
    CNode * p = pHead;              /*先把结点对象指针指向根结点*/
    for(int i=1; i<=length; i++)
    {                               /*循环整个链表
                                    判断是否是有效指针*/
        if(p!=NULL)
        {
        cout<<p->data<<",";         /*输出当前结点中的数据*/
        p=p->pNext;                 /*把当前结点移到下一结点*/
        }
        if(i%10==0)                 /*当等于10个数时,输出换行*/
        {
            cout<<endl;
```

图 3.1　多行注释

（2）使用“//”，从“//”开始直到所在行的行尾，所有字符都被当作注释处理。适用于单行注释信息。这种方法已经遇到过多次，如图 3.2 所示。

```
    length = 0;
}
CList::CList(int n)
{
    pHead = new CNode(n);
    length = 1;
}
void CList::insert(int n)
{
    CNode * p = new CNode(n);       //创建新结点对象
    p->pNext = pHead;               //把新结点的指向下一结点的指针指向根结点
    pHead = p;                      //把新结点变成要根结点
    length++;
}
void CList::remove()
{
    if(length > 0)
    {
        CNode * p = pHead;          //把结点p指针指向链表根结点
        pHead = pHead->pNext;       // head指向链首结点的下一个结点
        delete p;                   //删除p指向的结点
        length--;                   //长度减1
```

图 3.2　单行注释

在真实的程序编程中，上面介绍的注释方法可以根据不同的情况进行选用，增强了注

释和阅读的灵活性。

🔔**注意：**好的注释，能够提高程序设计或者代码阅读的效率。

3.3　C++中的常用数据类型

在 C++程序中数据的类型分为很多种，这就好比人的个头有高低、体重有胖瘦、皮肤有黑白一样。不同类型的数据，在数据存放的空间和处理数据的时间上都是各不相同的。本节的主要内容就是介绍 C++中常用的基本数据类型、声明关键字和格式。

🔔**注意：**在游戏设计中对于数据的类型尤为重视。因为采用的数据类型不同，游戏最后的运行速度和处理效果是有质的差别的。

3.3.1　整数型数据

所谓的整数型数据类是指数据是没有小数部分的，其值可以是正，也可以是负，简称为整数型。例如日常生活中用到的 12、–22、1234、2009、–999、–2009、0 等都是整数型的。其中 12、1234、2009 等又被称为正整数，0 一般也被当作正整数处理。而–22、–999，–2009 等被称为负整数。在 C++中用 int 这个关键字来声明一个存放整数型数据的变量（变量这个概念将在 3.4 节中进行讲解）。例如：

```
int nCount;                          //声明一个整数型变量 nCount
int nNum;                            //声明一个整数型变量 nNum
int abc;.                            //声明一个整数型变量 abc
```

声明整数型时要进行初始化，其方法如下：

```
int nCount=100;
```

🔔**说明：**在 C++程序中所有数据类型都是通过关键字来声明的。

知道了如何声明一个整数型变量，那么一个整数型在计算机中可以表示的数值的大小范围是多少呢？如表 3.2 所示。

表 3.2　有符号整数范围

计算机系统	最　大　值	最　小　值
16 位机	32 767	–32 768
32 位机	2 147 483 647	–2 147 483 648

🔔**注意：**整数变量能存储的最大值，是由计算机给其分配的存储空间的大小决定的。在不同的计算机系统中，其表示的范围是不同的。例如，早期的 16 位计算机是使用 2 个字节，而 32 位计算机则是使用 4 个字节。一般，现在的计算机都是 32 位的，不过也有 64 位的。

虽然整数型能表示的数已经比较大，但也有不够用的情况。例如，在有的游戏中表示银河系两个星球之间的距离是多少千米，或者太阳系的直径是多少米等数据时，整数型变量就不够了。为此 C++中就引入了长整数类型。用 long 这个关键字来声明一个长整数型变量。例如：

```
long lLen;                                    //声明长整数型变量 iLen
long lCount;                                  //声明长整数型变量 iCount
long lTime;                                   //声明长整数型变量 iTime
```

长整数型变量，其存储数值的范围如表 3.3 所示。

表 3.3　长整数范围

计算机系统	最　大　值	最　小　值
16 位机	2 147 483 647	–2 147 483 648
32 位机	2 147 483 647	–2 147 483 648

从上面的表中会发现一个问题，长整数和整数其实只是在 16 位计算机上有表示范围的差异，而在 32 位计算机上是无差异的。这是因为在 32 位计算机上，一般把长整数和整数的表示范围设置为相同大小。

在计算机中长整数可能会降低程序执行的速度，所以有时候为了节约空间和提高程序执行速度，C++中又引入了短整数。在 C++中用 short 这个关键字来声明一个短整数型变量，其数值的范围在 16 位和 32 位计算机上都是一样的，为 32 767～–32 768。例如：

```
short sCount;                                 //声明短整数型变量 sCount
short sNum;                                   //声明短整数型变量 sNum
short sLen;                                   //声明短整数型变量 sLen
```

其实，在日常生活中大家一般使用的都是正整数，即数学中的自然数，负数是比较少使用的。所以计算机为了在不增加存储空间和不影响执行速度的前提下，在存储数据时，直接把存储数据的符号去掉，这样变量能表示的最大值就扩大了，这种存储的数据在 C++中就被称为无符号数据类型。

C++中声明无符号整数型变量，是在 int 关键字前加上 unsigned 关键字。例如：

```
unsigned int unCount;                         //声明无符号整数型变量 unCount
unsigned int unNum;                           //声明无符号整数型变量 unNum
unsigned int abc;                             //声明无符号整数型变量 abc
```

既然是把最大值扩大了，那么其数值范围也不同了，无符号整数型变量存放的数值范围如表 3.4 所示。

表 3.4　无符号整数范围

计算机系统	最　大　值	最　小　值
16 位机	65 535	0
32 位机	4 294 967 295	0

既然有无符号的整数型变量，那么也就有无符号的长整数和短整数型变量，其声明都是在原关键字前加上 unsigned 关键字。例如：

```
unsigned long ulCount;              //声明无符号的长整数型变量 uiCount
unsigned long ulNum;                //声明无符号的长整数型变量 uiNum
unsigned short usLen;               //声明无符号的短整数型变量 usLen
unsigned short usCount;             //声明无符号的短整数型变量 usCount
```

无符号短整数其数值范围在 16 位和 32 位计算机上都是 65 535～0。而无符号的长整数其数值范围如表 3.5 所示。

表 3.5　无符号长整数范围

计算机系统	最　大　值	最　小　值
16 位机	4 294 967 295	0
32 位机	4 294 967 295	0

3.3.2　实数型数据

实数型数据是由一个整数部分、一个小数部分以及可选的后缀组成的。例如，3.1415、0.5、0.875 等都是实数型数据。在 C++中把实数型数据按照其表示的数值范围进行分类，可分为单精度、双精度和长双精度 3 种数据类型。

（1）声明一个单精度的实数变量使用 float 关键字，所以单精度的实数型又被称为浮点型。例如：

```
float half;                         //声明单精度的实数变量 half
float pi;                           //声明单精度的实数变量 pi
float num;                          //声明单精度的实数变量 num
```

单精度实数类型一般是以 32 位进行存储表示的，其数值表示范围为 3.4e+38～3.4e–38，最多只提供 7 位有效数字。

（2）双精度数据类型和单精度数据类似，因为其要占据的存储空间是单精度数据类型的两倍，所以被称为双精度数据类型。声明一个双精度类型的实数变量，使用 double 关键字。例如：

```
double dbNum1;                      //声明双精度的实数变量 dbNum1
double dbNum2;                      //声明双精度的实数变量 dbNum2
double dbNum3;                      //声明双精度的实数变量 dbNum3
```

双精度实数类型是以 64 位进行存储表示的，其数值范围为 1.7E–308～1.7E+308，最多可提供 16 位的有效数字。

（3）长双精度数据类型，理论上是双精度占用存储空间的两倍，但在实际中还是只占用 64 位的存储空间。声明一个长双精度实数变量，使用 long double 关键字。例如：

```
long double dbNum1;                 //声明长双精度的实数变量 dbNum1
long double dbNum2;                 //声明长双精度的实数变量 dbNum2
long double dbNum3;                 //声明长双精度的实数变量 dbNum3
```

长双精度因为其占用的存储空间和双精度相同，所以其实数类型的数值表示范围还是为 1.7E–308～1.7E+308，最多可提供 16 位有效数字。

实数在 C++中也可以包含一个指数形式的数值，其类似于数字的科学计数表示法。

例如：

```
1.5e3 其中 1.5 是尾数部分，3 是指数部分，表示的数值为 1500。
1.23e-2 其中 1.23 是尾数部分，-2 是指数部分，表示的数值为 0.0123
```

注意：字母 e 或 E 之前（即尾数部分）必须是有数字的。e 或 E 后面的指数部分必须是整数。

所有的实数型变量在声明时，对其进行初始化方法如下：

```
float half = 11111.511;
double dwSize = 11111.105;
```

由于 float 型的变量只能存储 4 个字节，有效数字为 7 位。所以其中 half 中的最后一位小数是不起作用的，即其值为 11111.51。但如果是 double 型的，则 dwSize 中是能够存储全部 11111.105 这个数的。对于这一点，读者只需要记住就行了，不必细究。

3.3.3　字符型数据

通常字符型数据变量是用来存储计算机中的字符的。例如：‘A’，‘#’，‘z’，‘1’等都是字符型的数据。在计算机中，字符型变量并不是直接存储字符，而是存储字符的 ASCII（American Standard Code for Information Interchange，美国标准信息交换码）码值。

ASCII 码是目前计算机中用得最广泛的字符集及编码，是由美国国家标准局（即 ANSI）制定的，其已被国际标准化组织（ISO）定为国际标准，称为 ISO 646 标准。适用于所有拉丁文字字母。

根据 ASCII 码标准，数值 65 代表大写的‘A’，而 97 则代表小写字母‘a’，表 3.6 列出了 ASCII 码表中前 128 个编码的十进制表示，读者可以不必记住这些代码值，但必须理解每个 ASCII 码都与一个特定的数值相对应的这个概念，ASCII 码表如表 3.6 所示。

表 3.6　ASCII码表

十进制	符　号	十进制	符　号	十进制	符　号	十进制	符　号
0	NUT	13	CR	26	SUB	39	,
1	SOH	14	SO	27	ESC	40	(
2	STX	15	SI	28	FS	41	)
3	ETX	16	DLE	29	GS	42	*
4	EOT	17	DCI	30	RS	43	+
5	ENQ	18	DC2	31	US	44	,
6	ACK	19	DC3	32	（space）	45	-
7	BEL	20	DC4	33	!	46	.
8	BS	21	NAK	34	”	47	/
9	HT	22	SYN	35	#	48	0
10	LF	23	TB	36	$	49	1
11	VT	24	CAN	37	%	50	2
12	FF	25	EM	38	&	51	3

续表

十进制	符　号	十进制	符　号	十进制	符　号	十进制	符　号
52	4	71	G	90	Z	109	m
53	5	72	H	91	[	110	n
54	6	73	I	92	\	111	o
55	7	74	J	93	]	112	p
56	8	75	K	94	^	113	q
57	9	76	L	95	—	114	r
58	:	77	M	96	、	115	s
59	;	78	N	97	a	116	t
60	<	79	O	98	b	117	u
61	=	80	P	99	c	118	v
62	>	81	Q	100	d	119	w
63	?	82	R	101	e	120	x
64	@	83	X	102	f	121	y
65	A	84	T	103	g	122	z
66	B	85	U	104	h	123	{
67	C	86	V	105	i	124	|
68	D	87	W	106	j	125	}
69	E	88	X	107	k	126	~
70	F	89	Y	108	l	127	DEL

看一下这张表，会发现其中有不少奇怪的字母组合，这些字母组合的含义如表 3.7 所示。

表 3.7　特殊字符的含义

NUL	空	VT	垂直制表	SYN	空转同步
SOH	标题开始	FF	走纸控制	ETB	信息组传送结束
STX	正文开始	CR	回车	CAN	作废
ETX	正文结束	SO	移位输出	EM	纸尽
EOY	传输结束	SI	移位输入	SUB	换置
ENQ	询问字符	DLE	空格	ESC	换码
ACK	承认	DC1	设备控制 1	FS	文字分隔符
BEL	报警	DC2	设备控制 2	GS	组分隔符
BS	退一格	DC3	设备控制 3	RS	记录分隔符
HT	横向列表	DC4	设备控制 4	US	单元分隔符
LF	换行	NAK	否定	DEL	删除

字符型在计算机中以 8 位进行存储，其表示数值的范围为 127～–128。在 C++中声明一个字符型的变量，使用关键字 char，例如：

```
char no;                                    //声明一个字符型变量 no
char yes;                                   //声明一个字符型变量 yes
```

```
char tmp;                              //声明一个字符型变量 tmp
```

字符型的数据都是使用单引号引起来，例如‘Y’。如果要在声明字符型变量时进行初始化，其方法如下。

```
char no='N';                           //声明字符变量 no，并初始化为 N
```

在 C++中还有一些是由单引号引起来的，但由多个字符所组成的特殊字符，C++把这些特殊字符当作单一字符来处理，称为转义字符。转义字符都具有特殊的功能，如表 3.8 所示。

表 3.8　转义字符

转 义 字 符	说　　明
\a	蜂鸣
\b	回退键
\f	换页
\n	换行
\r	回车换行
\t	水平制表
\v	垂直制表
\\	反斜杠
\’	单引号
\”	双引号
\?	问号
\nnn	八进制位模式，nnn 是一个八进制数
\xnn	十六进制位模式，xnn 是一个十六进制数

前面在讲解整数型时，知道分为“有符号”和“无符号”两种。其实字符型数据也是如此。如果要声明一个无符号的字符型数据，只需要在 char 关键字的前面加上 unsigned 关键字，其表示的数值范围就变为 255～0。例如：

```
usigned char no;                       //声明无符号字符型变量 no
usigned char id;                       //声明无符号字符型变量 id
```

现在再来了解另一种字符串型数据，所谓字符串就是由 0 个或多个字符组成的有限序列。例如 abcde、Hello、1234 这些都是字符串。字符串就相当于一个字符数组。

注意： *在表示单独字符的时候，使用单引号，而在表示字符串或多于一个字符的时候必须使用双引号。*

3.3.4　布尔型数据

布尔型数据即是逻辑型的简单数据值，其只有包含两个值：false（假）和 true（真），前者序号为 0，后者序号为 1。在 C++中，虽然布尔类型的数据量最少，但用途却非常广泛，其主要用于程序设计中的流程控制和逻辑判断。

声明一个布尔型的变量，使用 bool 关键字，例如：

```
bool bFlg;                          //声明一个布尔型变量 bFlg
bool bOpen;                         //声明一个布尔型变量 bOpen
bool bClose;                        //声明一个布尔型变量 bClose
```

在声明布尔型的变量时，也可以进行初始化，其方法如下。

```
bool no = false;                    //在定义布尔型变量 no 时，初始化为 false
```

注意：在 C++中，只有 0 才能表示为假。其他任何数值代表布尔型变量时都表示真，包括负数。

总之，在设计游戏需要的数据类型时，应尽量满足游戏设计的要求。比如要计算有小数参与的运算，就应该选择实数类型，而不能选择整数型。而在没有小数参与的运算时，就应该选择整数型。

3.4　C++中的常量与变量

前面的数据类型一节中，常常会提到“变量”，在本节将会对其进行详细的说明。不仅如此，还将学习到另一个概念——“常量”。

3.4.1　变量的定义

变量就是在内存中，可以不断地被程序操作的、有名字的存储区。在代码中可以只使用一个变量，也可以使用多个变量。在使用变量前，必须先要创建一个变量。创建变量的 C++语句称为变量的声明，在前面已经见过。其格式如下：

```
变量数据类型　变量名;
int nLen;                           //声明一个整数型的变量 nLen
```

也可以使用如下格式同时声明 n 个同类型的变量，但要注意一定是同类型的才能如此。

```
变量数据类型　变量名 1,变量名 2,…,变量名 n;
int a,b,c;                          //同时声明三个整数型变量 a,b,c
```

不同类型的变量不能在同一语句中进行声明，下面的方式是错误的。例如：

```
int  len,short  count;              //在同一行声明不同类型变量，这是错误的
```

通过前面的学习，知道了 C++中的每一个变量都有特定的类型，该类型决定了变量的内存大小和布局，以及能够存储于该内存中的值的取值范围和可应用于该变量上的操作集。这样计算机就可以正确理解变量中的数据含义了。

在前面的声明格式中看到的“变量名”就是变量的名字，就像每个人都有自己的名字一样，给变量起名的意义是为了区分不同的变量。在 C++中的变量名称是不能随便取的，除了必须遵循标识符命名规则外，还应该尽量遵循下面几条编程经验，虽然不是必须遵循

的，但为了方便对代码的阅读和理解，在程序设计时应尽量采用。

- 见名见意，指当看到这个变量的名称时就能望名知意，这样能便于读者阅读和进行程序设计。例如，用 count 来作为计数器变量名，用 len 作为长度变量名，用 age 作为年龄变量名等。
- 尽量不用汉语拼音。例如，声明一个姓名变量，应该使用 name 为变量名，而不是用 xingming 作为变量名。
- 命名不宜过长。用很长一串字符来命名会减慢编程的速度，也会使阅读程序困难。不要用过长的名称来命名变量，一般采用缩写来命名，例如，用 init 来表示初始化等。
- 采用驼峰标记法和匈牙利标记法来命名变量。

声明变量之后必须对其进行初始化才能使用，否则会导致程序运算错误。对变量进行初始化后，变量的值才会有效。

```
变量名=初始值;
```

也可以在声明变量的时候进行初始化，其格式如下：

```
变量类型 变量名=初始值;
```

要注意，在变量初始化的时候，设置的初始值一定要符合变量的数据类型。例如：

```
int nPI = 3.1415                    //错误的初始化
int nCount = 100                    //正确的初始化
```

3.4.2　常量的定义

与变量相对的就是常量。变量的值可以在程序中随时而改变，而常量的值是不能被改变的，所以其被称为常量。比如在游戏设计中，设置常常会把圆周率 π 设置为常量。其值就默认等于 3.141 592 7。

在 C++中常量分为如下两种：

- 文字常量，例如整数 888、字母 b 等都是文学常量。
- 自定义常量，即自己声明的常量。

自定义常量的声明格式与变量声明差不多，只是在语句的最前面多了一个 const 关键字来说明这是一个常量。例如：

```
const 数据类型 常量名=文字常量;
const int MAX=1000;                 //声明一个 int 型的常量 MAX，其值为 1000
const double PI = 3.1415927f;       //声明一个 float 型的常量
```

常量必须在声明时进行初始化，并且在除声明语句外，在程序的任何地方不能再对其进行赋值。这是和变量不同的地方之一。

声明一个单精度实型常量，需要该文字常量的最后加上 F 或者 f；如果要表示长双精度实型常量，则要在该文字常量的最后加上 L 或者 l。例如：

```
1.05f;                          //声明的是单精度实型常量
```

```
1.75l                                    //声明的是长双精度实型常量
2.55F;                                   //因为采用了 F 说明，所以是单精度常量
3.75L;                                   //因为采用了 L 说明，所以是长双精度常量
```

事实上，只要在游戏程序中不需要改变的值，都应用常量来表示。这样做可以提高程序稳定性和严谨性。

3.5　C++中的运算符与表达式

除了变量和常量外，在 C++中还有其自身支持的、操作变量的运算符和表达式。运算符是表示对数值进行一种运算的符号。其对操作数进行运算，操作数可以是一个数值、变量或者常量。比如，大家平时接触到的数学中的运算符（+、–）及比较运算符（>、<、=）等。这些在 C++中也是支持的。

C++的运算符范围很广，有带一个操作数的，也有带两个操作数的；有些是专用的，而有些是由其他运算符派生出来的；甚至有些是重载的（将在后面的章节进行讲解）。所以很难找到一个程序能把所有的运算符都用上。

表达式是与运算符对应的，其代表的是由运算符和操作数组成的式子。由于运算符很丰富，因此表达式的种类也很多。最简单的表达式是常量或者变量。

在本章中，笔者只列举出常用的几种简单的运算符和表达式。同时在示例代码中给出其使用方法。

3.5.1　赋值运算符

在程序设计中最常用到的赋值运算符。C++规定赋值运算符为=，其作用是将一个数据赋值给一个变量，类似于数学中的等于符号。如 num1=1 的作用是把字符常量“1”赋值给变量 num1。也可以将一个表达式的值赋值给一个变量，由赋值运算符将一个变量和一个表达式连接起来的语句被称为“赋值表达式”，格式为：

```
左值=表达式;
```

赋值运算符的操作过程如下：

（1）计算右边表达式的值。

（2）把右边计算出来的值，存放在左值之中。

左值可以理解为变量或者声明语句中的自定义常量。例如：

```
a = 11;                                  //赋值给左值变量 a，值为 11
const int A=10;                          //赋值给左值常量 A，值为 10
```

3.5.2　算术运算符

在 C++中支持的运算符如表 3.9 所示。

表 3.9 算术运算符

运 算 符	说 明	例 子
+	加法运算符，或者表示正值	2+3 +8
–	减法运算符，或者表示负值	5–2 –88
*	乘法运算符	7*2
/	除法运算符	9/3
%	取模运算符，又称取余运算符	10%3 的结果是 1

算术运算符的计算方法同数学中的基本一致。不过要注意：当除号两边的数为整数型数据时，其结果为整数。如 5/3 的结果值为 1，舍去小数部分。但是如果除数或者被除数中有一个为负值，则舍去的方向是不固定的。例如，–5/3 有的电脑上–1，有的是–2。

下面的例子说明了算术运算符的使用方法，如代码 3.1 所示【代码参考：光盘的源代码\C03\Demo1.dsp】

代码 3.1 算术与赋值运算符示例

```
01  #include <windows.h>                  //一个 Windows 应用程序应该包含的头文件
02  #include <stdio.h>                    //标准输入输出流文件
03  LRESULT CALLBACK WinSunProc           /*声明一个回调函数*/
04  (
05   HWND hwnd,                           /*窗口的句柄*/
06   UINT uMsg,                           /*窗口的消息*/
07   WPARAM wParam,
08   LPARAM lParam
09   );
10  int WINAPI WinMain
11  (
12   HINSTANCE hInstance,                 //实例句柄，当前应用程序的实例句柄
13   HINSTANCE hPrevInstance,             //默认这个参数为 NULL
14   LPSTR lpCmdLine,                     //储存一个命令行参数
15   int nCmdShow)
16  {
17      WNDCLASS wndcls;                  //定义一个窗口对象
18      wndcls.cbClsExtra=0;              //指定额外内存空间
19      wndcls.cbWndExtra=0;              //指定额外内存空间
20                                        //指定窗口背景色
21      wndcls.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
22                                        //设置光标样式
23      wndcls.hCursor=LoadCursor(NULL,IDC_CROSS);
24                                        //设置图标样式
25      wndcls.hIcon=LoadIcon(NULL,IDI_ERROR);
26      wndcls.hInstance=hInstance;       //指定窗口实例句柄
27      wndcls.lpfnWndProc=WinSunProc;    //指定窗口函数，即窗口主处理函数
28      //窗口类名称
29      wndcls.lpszClassName="Visual C++ Game";
30      wndcls.lpszMenuName=NULL;         //菜单
31      wndcls.style= CS_HREDRAW|CS_VREDRAW;
32      RegisterClass(&wndcls);           //注册窗口类
33      HWND hwnd;                        //声明窗口句柄
34      hwnd=CreateWindow                 /*创建窗口，但这里的窗口是不会显示的*/
35          (
36          "Visual C++ Game",            /*已注册窗口类的名称*/
37          "Visual C++ 游戏开发",        /*窗口标题*/
```

```
38          WS_OVERLAPPEDWINDOW,          /*窗口风格*/
39          200,                          /*窗口位置的横坐标*/
40          200,                          /*窗口位置的纵坐标*/
41          600,                          /*窗口的宽度*/
42          400,                          /*窗口的高度*/
43          NULL,
44          NULL,
45          hInstance,                    //实例句柄
46          NULL);
47      //在这里真正显示窗口
48      ShowWindow(hwnd,SW_SHOWNORMAL);
49      UpdateWindow(hwnd);               //更新显示
50      MSG msg;
51      while(GetMessage(&msg,NULL,0,0))
52      {
53          TranslateMessage(&msg);       //转换键盘消息
54          DispatchMessage(&msg);        //分派消息
55      }
56      return 0;
57  }
58  LRESULT CALLBACK WinSunProc(
59                                HWND hwnd,             /*窗口句柄*/
60                                UINT uMsg,             /*消息*/
61                                WPARAM wParam,         /*参数 1*/
62                                LPARAM lParam          /*参数 2*/
63                                )
64  {
65      char tmsg[128] = {0};
66      int num1, num2, num3, num4,num5;    //声明 5 个变量
67      num1 = 3+8;                         //加法运算
68      num2 = 10-7;                        //减法运算
69      num3 = 100*33;                      //乘法运算
70      num4 = 155/5;                       //除法运算
71      num5 = 9%2;                         //取模运算
72                                          //把运算符和结果输出到 tmsg 中
73      sprintf(tmsg, "3+8=%d 10-7=%d 100*33=%d 155/5=%d 9%%2=%d",
74              num1, num2, num3, num4, num5);
75      switch(uMsg)                        /*判断消息类型*/
76      {
77      case WM_PAINT:                      /*更新窗口消息*/
78          HDC hDC;                        /*定义 DC 设备*/
79          PAINTSTRUCT ps;
80          hDC=BeginPaint(hwnd,&ps);       /*得到设备 hDC*/
81          TextOut(hDC,150,0,tmsg,strlen(tmsg));
82          EndPaint(hwnd,&ps);
83          break;
84      case WM_CLOSE:                /*当单击“关闭”按钮时，产生关闭消息*/
85          if(IDYES==MessageBox(hwnd,"是否真的结束？","游戏开发",MB_
            YESNO))
86          {
87              DestroyWindow(hwnd);      /*单击“确定”按钮，销毁窗口*/
88          }
89          break;
90      case WM_DESTROY:                  /*销毁窗口消息*/
91          PostQuitMessage(0);           /*退出程序*/
92          break;
```

```
93      default:
94          //在 default:处必须调用 DefWindowProc，这是 Windows 内部默认的消息处
            理函数
95          return DefWindowProc(hwnd,uMsg,wParam,lParam);
96      }
97      return 0;
98  }
```

代码解析：第 71 行，是用%运算符来表示这是一个取模运算。其意思是用来得到除法运算后的余数，即数学中的取余运算。其方法是先用 9 除以 2，得到商是 4，余数是 1，再把商丢弃，最后得到余数，并保存到变量 num5 中。第 73 行是把变量的结果格式化输出到字符数组 tmsg 中，格式化输出的方法是属于 C 语言的内容，请参见相关书籍，这里不再介绍。代码编译运行的结果如图 3.3 所示。

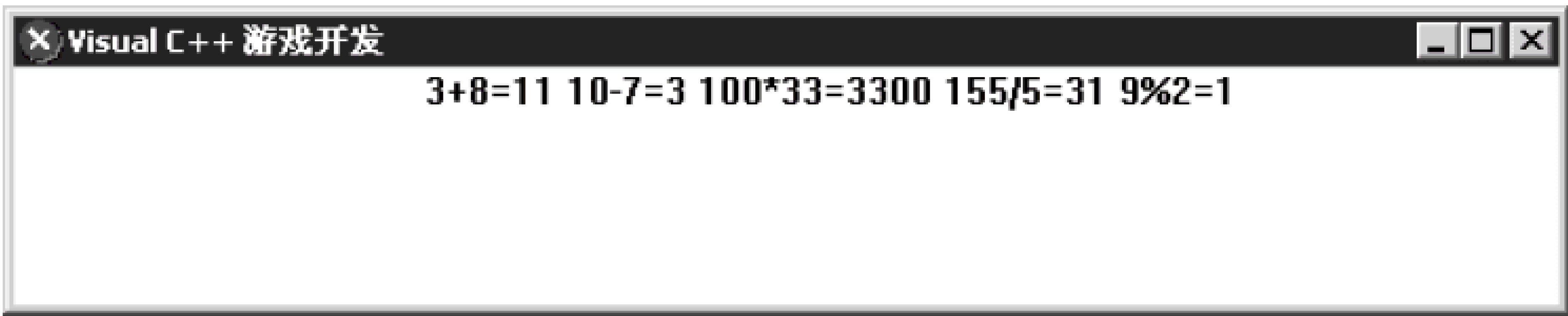


图 3.3　赋值与算术运算符示例运行结果

在 C++中用算术运算符和括号将运算对象连接起来的、符合 C++语法规则的语句，被称为算术表达式。运算对象包含常量、变量等。如代码 3.1 中的 3+8、10–7、num2*num1、num3/5，以及 num3%2 这些语句都是算术表达式。

3.5.3　自增与自减运算符

有两个特殊的运算符++和– –，在 C++中被称为自增和自减运算符，运算符++的作用是使变量的值增加 1。其表达式如下：

```
a++;            //相当于 a=a+1
++a;            //相当于 a=a+1
```

a++和++a 的作用相当于 a=a+1 都是将 a 的值加 1，但也有些不同。因为 a++是先使用 a 的值，再执行 a=a+1。而++a 是先执行 a=a+1，然后使用 a 的值。如果使用另一个变量来存放运算结果，这两种形式的不同就容易理解了。

例如：现在假定 a 的初值是 30。

```
int a=30;
int b=a++;
```

上面的语句是先将 a 的值 30 赋值给 b，然后 a 增加 1。最后的结果是变量 b 的值为 30，而变量 a 的值为 31。如果把语句修改为：

```
Int a=30;
int b=++a;
```

上面的语句是先将 a 增加 1，然后将 a 的新值 31 放在 b 中，最后的变量结果是变量 b 的值为 31，而变量 a 的值仍为 31。

在这里，把++a 念为“a 先加”而把 a++念为“a 后加”。好了，相信现在大家应该明

白++的作用了吧，其实另外一个运算符––正好和++的作用相反，是将变量的值减少 1。但运算过程是一样的，这里不再复述。

自增（++）和自减（––）运行符，只能用于变量，而不能用于常量或者表达式，如“10++”或者（a*c）++都是不合法的。因为 10 是常量，常量的值不能被改变。而“（a*c）++”是一个表达式，相加之后的结果没有变量可供存放。

3.5.4　复合运算符

复合运算符是对赋值运算符的扩展，其是在“=”赋值符之前加上其他运算符，就构成了复合运算符。使用复合运算符可以简化程序，使程序看上去精练，提高代码的编译效率。其表达式格式如下：

```
变量　运算符=表达式;
```

例如：

```
b+=4;         //等价于 b=b+4
b-=4;         //等价于 b=b-2
```

以“b+=4”为例来说明，其相当于先使变量 b 加 4，然后再重新赋值给变量 b。同理，“b–=4”是先使变量 b 减 4，然后再赋值给变量 b。为了便于记忆，读者也可以这样理解：

（1）a+=b，其中 a 为变量，而 b 为表达式。

（2）把 a+移动到=的右侧，变成=a+b。

（3）在=的左边补上变量名，变成 a=a+b。

即变成如下格式：

```
变量 = 变量 运算符 表达式
```

如果表达式是由几个表达式组成的，则相当于该表达式是有括号的。例如：

- a*=3*（2+3）。
- =a*（3*（2+3））。
- a=a*（3*（2+3）），不要写成 a=a*3*（2+3）。

凡是只有两个操作数的运算符，都可以与赋值运算符组成复合运算符。在 C++中规定了如下所示两类共 10 种复合运算符。

- 算术类：+=、– =、*=、/=、%=。
- 位运算类：<<=、>>=、&=、^=、|=。

3.5.5　位运算符

在计算机内所有的数据，总是用 0 和 1 的组合进行存储的，也就是二进制。而 8 个二进制位构成一个字节，两个字节构成一个字，也就是 16 位。有时按位来操作数据是很必要的，程序员可以通过改变存储在位、字节或者字中的 0 和 1 来改变它的值，这也就是位操作的由来。其表达式如下：

变量 位运算符 变量或者常量

1．左位移运算符（<<）

左位移运算符是将一个字中的数据位左移指定的位数。左移位一次相当于对当前这个数值每次乘 2。其实现的方法如下：

（1）在 32 位计算机中，7 被表示为 0000 0000 0000 0111。

（2）左移 4 位，变成 0000 0000 0111 …。

（3）空位用 0 来填充，变成 0000 0000 0111 0000。

（4）这个二进制数的数值正好是 112。

2．右位移运算符（>>）

右位移运算符刚好和左位移相反，它是将一个字中的数据位右移指定的位数。实现方法如下：

（1）在 32 位计算机中，128 被表示为 0000 0000 1000 0000。

（2）右移 2 位，变成…0 0000 0001 0000。

（3）空位用 0 来填充，变成 0000 0000 0001 0000。

（4）这个二进制数的数值正好是 16。

不过要注意，在右移位时，只有移位的变量是正数时，才会用 0 作为填充位，而如果变量是负数的话，右移位的结果是无法预料的，左移位无此限制。右移位一次相当于对当前数值每次除 2。

3．按位与运算符（&）

按位与运算符（&）是用来得到两个整数操作数的逻辑乘积。当被“与”的两位是 1 时结果是 1，否则结果为 0，如表 3.10 所示。

表 3.10　按位与示例表

操作位 1	操作位 2	结　　果
0	0	0
0	1	0
1	0	0
1	1	1

现在设定 n 的初值为 250，m 的初值为 86。那么这两个数进行按位与运算后，其结果如表 3.11 所示。

表 3.11　按位与运算结果

数　　值	说　　明
1111 1010	250 二进制
0101 0110	86 二进制
0101 0010	对齐后，根据表 5.2 方法竖式运算，结果为 82

4. 按位或运算符（|）

按位或运算符（|）是用来得到两个整数操作数的逻辑和。当被“或”的两位是 0 时结果是 0，否则结果为 1，如表 3.12 所示。

表 3.12　按位或示例表

操作位 1	操作位 2	结　　果
0	0	0
0	1	1
1	0	1
1	1	1

现在设定 n 的初值为 250，m 的初值为 86，进行按位或运算过程和结果如表 3.13 所示。

表 3.13　按位或运算结果

数　　值	说　　明
1111 1010	250 二进制
0101 0110	86 二进制
1111 1110	对齐后，根据表 5.4 方法竖式运算，结果为 254

5. 按位异或运算符（^）

按位异或运算符（^）是用来得到两位操作数之间的异或结果的，其示例见表 3.14。

表 3.14　按位异或示例表

操作位 1	操作位 2	结　　果
0	0	0
0	1	1
1	0	1
1	1	0

现在设定 n 的初值为 250，m 的初值为 86，进行按位异或运算过程和结果如表 3.15 所示。

表 3.15　按位异或运算结果

数　　值	说　　明
1111 1010	250 二进制
0101 0110	86 二进制
1010 1100	对齐后，根据表 5.4 方法竖式运算，结果为 172

3.5.6　关系运算符

关系运算符与数学的比较运算符的运算方法相同，但标记格式不同。C++提供了 6 种

关系运算符，如表 3.16 所示。

表 3.16　关系运算符

运　算　符	说　　明	例　　子
<	小于	a<10
<=	小于等于	a<=10
>	大于	b>10
>=	大于等于	b>=200
==	等于	a==b
!=	不等于	a!=b

用关系运算符将两个表达式连接起来的式子，称为关系表达式，例如：

```
a>b                                  //a 大于 b
a+b<b+c                              //a+b 小于 b+c
(a=30)<(b=50)                        //变量 a 的值 30 小于变量 b 的值 50
b==a                                 //变量 b 和变量 a 相等
```

所有关系表达式的值是一个布尔值，即“真”（1）或者“假”（0）。例如，关系表达式“5==3”的值为“假”，而“5>3”的值为“真”。

3.6　C++中的控制语句

一个结构化程序设计需要 3 种基本结构：**连续结构、选择结构和循环结构**。在 C++中这 3 种结构分别对应基本语句、条件选择语句和循环语句。掌握好这 3 种语句的使用，对于程序设计是非常重要的。

和其他高级语言一样，C++的语句也是用来向计算机系统发出操作指令的。这就好像操练时的“向左转”、“向右转”、“稍息”、“立正”等一系列口令（即语句）。有了这些口令大家才能走出整齐的队列。

3.6.1　基本语句

一条基本语句经过编译后，将产生几条机器指令。为实现特定功能的程序一般又包含若干条基本语句。通常把 C++的基本语句分为如下两种。

1．简单的基础语句

在一个 C++程序中有许多表达式。有由算术运算符组成的算术表达式、由赋值运算符组成的赋值表达式、由位运算符组成的位运算表达式等。例如：

```
x=x+4                      //算术表达式
x=7*y                      //赋值表达式
y=4,x=3,n/55
x++,y--
x|y  +n^m                  //位运算表达式
```

以上这些都是表达式，前面已经接触过了。现在要学习表达式语句。读者是不是被搞糊涂了，前面不是已经学习过了吗？其实不然，表达式与表达式语句的不同之处在于“C++中任意一个表达式必须加上一个分号（;）才能成为一个表达式语句”。

以下才是真正的表达式语句，而不是表达式。

```
x=x+4;                    //赋值语句
x=7*y;
y=4,x=3,n/55;             //赋值语句
x++;y--;                  //自增和自减语句
x|y+n&m;                  //位运算语句
```

在表达式语句中，还有一种特殊的语句，被称为空语句。是指只有一个分号（;）而不包含表达式的语句。空语句是一种不做任何操作的语句。该语句主要用在一些需要一条语句，但又不作任何操作的地方。例如，有的循环语句中的循环体就是使用空语句来代替原有语句的。

2．复合语句

所谓复合语句就是指由两条或者两条以上的语句组成，并由一对花括号（{}）包含起来的语句。但在语法意义上其相当于一条语句，所以又被称为块语句。含有一条或者多条说明的复合语句称为分程序，也叫块结构。如代码 3.2 就是由复合语句组成的分程序。

代码 3.2　分程序代码示例

```
{                 //块语句开始
    int i = 10;
    int n = 90;
    n=i+n;
    ...           //省略其他语句
}                 //块语句结束
```

3.6.2　条件选择语句

相信读者都遇到过这样一种情况：到一个风景区去旅游，如果路途比较远，就可以选择坐飞机，如果路途近，就可以选择坐汽车。根据给定的条件，即路途的远近来选择所走的路线。这就一种条件选择。在 C++中也提供了一种条件选择语句，其具有一定的判断能力，根据给定的条件来决定执行那些语句，不执行哪些语句。

1．if 语句

C++中使用 if 关键字来定义条件选择语句。所谓条件选择语句，就是指根据条件的不同可以进行不同的选择。比如平常生活中说的“如果今天下班时间早，就自己在家做饭吃，否则就在外面吃”。这句话是根据下班时间早晚的这个条件，来选择在家自己做饭吃，还是在外面吃。这就构成了一个简单的条件选择语句。在 C++中定义简单 if 条件选择语句的格式如下：

```
if （条件） 语句
```

其中，if 是关键字。“条件”是作为判断条件使用的各种表达式。一般是关系表达式或者后面将讲解的逻辑表达式，不过也可以是其他任意数据类型的变量或者常量（包括整数、实数、字符型和布尔型）。“语句”可以是单一语句，也可以是复合语句。例如：

```
if (5>3) cout<<"真"<<endl;                //单一语句
if(bflg == TRUE){                         //复合语句
    int n=10;
    int m=100;
    ...
}
```

这种 if 语句的执行流程如图 3.4 所示。

除了用 if 定义外，还可以使用 if（如果）和 else（否则）组合起来定义。格式如下：

```
if (条件) 语句1 else 语句2
```

根据“条件”中的表达式进行判断，如果值是真（非 0），则执行“语句 1”。执行完毕后，执行条件选择语句后的其他语句。如果值是假（0），则执行“语句 2”。例如：

```
if (x>y){                                 //if 语句
    nCount = 10;
}
else{                                     //else 语句
    nCount = 100;
}
```

这种 if-else 格式的语句执行流程如图 3.5 所示。

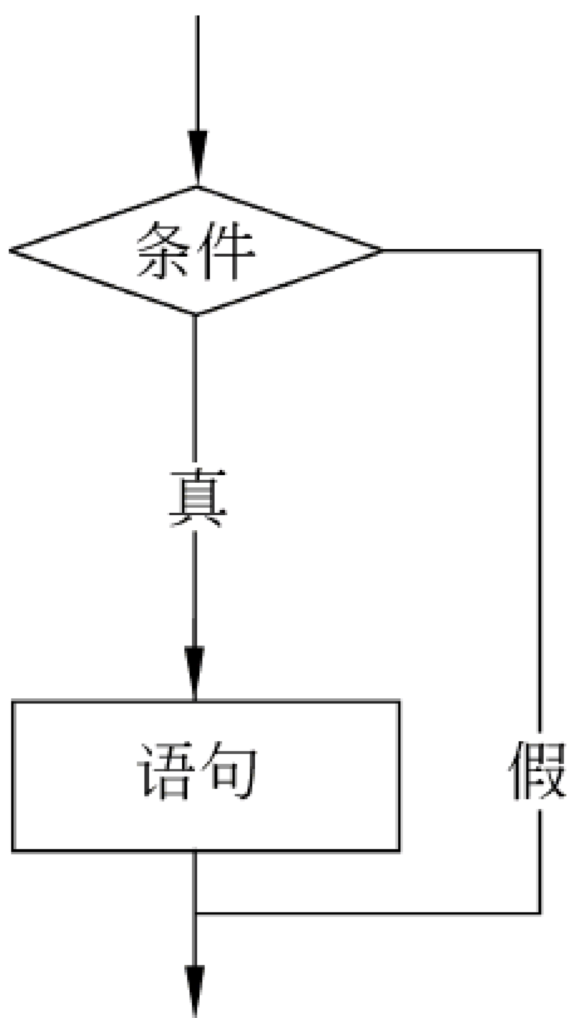


图 3.4　if 语句执行流程

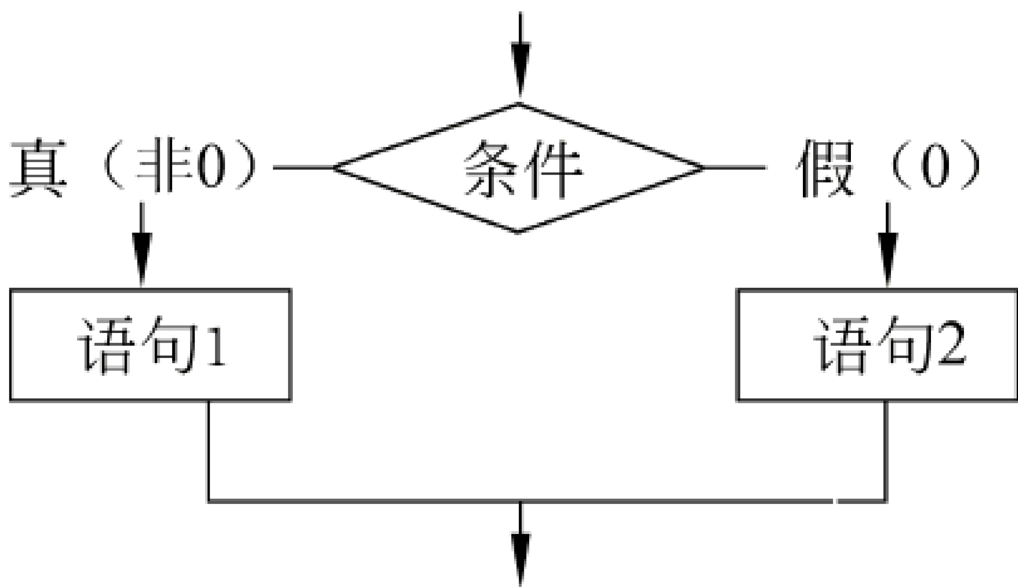


图 3.5　if-else 执行流程

除了上面两种外，还有一种比较复杂的格式，如下所示。

```
if (条件1){
    语句1
}
else if(条件2){
    语句2
}
else if(条件3){
    语句3
}
else if(条件n-1){
    语句n-1
```

```
}
else{
    语句 n
}
```

首先判断“条件 1”给出的表达式的值。如果该值为非 0，则执行“语句 1”，执行完毕后，转到条件选择语句后面继续执行其他的语句。如果该值为 0，则继续判断“条件 2”给出的表达式的值。

如果“条件 2”的值为非 0，则执行“语句 2”，执行完毕后，转到条件选择语句后面继续执行其他的语句。如果“条件 2”的值是 0，则继续判断“条件 3”给出的表达式的值，依次类推。如果所有条件表达式中的值都是 0，则执行 else 后的“语句 n+1”。如果没有 else，则什么也不做，转到条件选择语句后面继续执行其他语句。例如：

```
if (x>1000){
    y=1000;
}
else if (x>500){
    y=500;
}
else if (x>400){
    y=400;
}
else if (x>200){
    y=200;
}
else{
    y=100;
}
```

if-else if 格式的语句执行流程如图 3.6 所示。

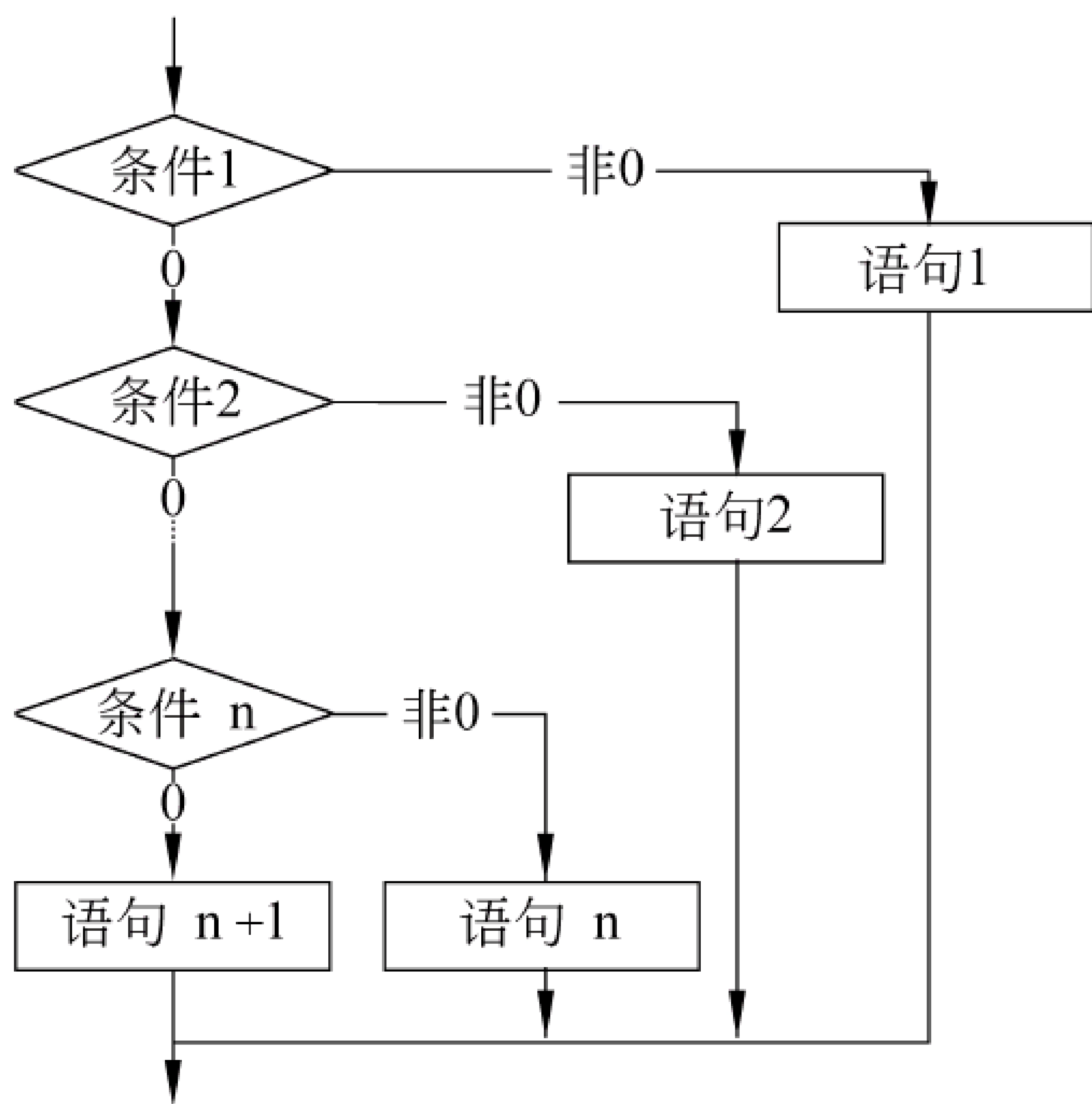


图 3.6　if-else if 执行流程

说明：在 C++中，所有的 if 语句都是可以嵌套的，if、if-else 或者 if-else if 中都可以再包含 if 语句。这里就不再给出，请读者自己实践。

2．switch 语句

在实际生活中，常常需要对很多数据进行逻辑分类判断。例如：学生的成绩分类（90 分以上的定为 A，80～89 分的定为 B，70～79 分的定为 C，60～69 分的定为 D，60 分以下的定为 E）；人口的年龄分类（老人、中年、青年、少年、儿童）；公司的工种分类等。对这些数据进行分类判断时，可以使用嵌套的 if-else if 语句来处理，但是这样做就会显得分支过多，而且嵌套的 if 语句层数很深，程序可读性就降低了。

为此 C++提供了 switch 语句来处理这样的问题，switch 语句是对一个表达式检查多个可能的常量值，是多分支选择语句。switch 语句的声明格式如下：

```
switch (条件表达式)
{
```

```
    case 常量表达式 1:语句 1
    case 常量表达式 2:语句 2
    case 常量表达式 3:语句 3
    ...
    case 常量表达式 n:语句 n
    default: 语句 n+1
}
```

其中，switch 是关键字，case 和 default 是子句关键字。常量表达式是指一个值为字符常量或者整数常量的表达式。语句可以是一条简单语句，可以是复合语句，也可以是空语句。例如：现在要将前面的代码 3.1 中 switch 语句用 if-else if 语句来代替，如代码 3.3 所示。

代码 3.3　算术与赋值运算符示例 if 语句型

```
01  #include <windows.h>                    //一个 Windows 应用程序应该包含的头文件
02  #include <stdio.h>                      //标准输入输出流文件
03  LRESULT CALLBACK WinSunProc             /*声明一个回调函数*/
04  (
05   HWND hwnd,                             /*窗口的句柄*/
06   UINT uMsg,                             /*窗口的消息*/
07   WPARAM wParam,
08   LPARAM lParam
09   );
10  int WINAPI WinMain
11  (
12   HINSTANCE hInstance,                   //实例句柄，当前应用程序的实例句柄
13   HINSTANCE hPrevInstance,               //默认这个参数为 NULL
14   LPSTR lpCmdLine,                       //储存一个命令行参数
15   int nCmdShow)
16  {
17      WNDCLASS wndcls;                    //定义一个窗口对象
18      wndcls.cbClsExtra=0;                //指定额外内存空间
19      wndcls.cbWndExtra=0;                //指定额外内存空间
20
21      wndcls.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
22                                          //指定窗口背景色
23      wndcls.hCursor=LoadCursor(NULL,IDC_CROSS);
24                                          //设置光标样式
25      wndcls.hIcon=LoadIcon(NULL,IDI_ERROR);      //设置图标样式
26      wndcls.hInstance=hInstance;         //指定窗口实例句柄
27      wndcls.lpfnWndProc=WinSunProc;      //指定窗口函数，即窗口主处理函数
28      //窗口类名称
29      wndcls.lpszClassName="Visual C++ Game";
30      wndcls.lpszMenuName=NULL;           //菜单
31      wndcls.style= CS_HREDRAW|CS_VREDRAW;
32      RegisterClass(&wndcls);             //注册窗口类
33      HWND hwnd;                          //声明窗口句柄
34      hwnd=CreateWindow                   /*创建窗口，但这里的窗口是不会显示的*/
35          (
36          "Visual C++ Game",              /*已注册窗口类的名称*/
37          "Visual C++ 游戏开发",          /*窗口标题*/
38          WS_OVERLAPPEDWINDOW,            /*窗口风格*/
39          200,                            /*窗口位置的横坐标*/
40          200,                            /*窗口位置的纵坐标*/
```

```
41          600,                          /*窗口的宽度*/
42          400,                          /*窗口的高度*/
43          NULL,
44          NULL,
45          hInstance,                    //实例句柄
46          NULL);
47      //在这里真正显示窗口
48      ShowWindow(hwnd,SW_SHOWNORMAL);
49      UpdateWindow(hwnd);               //更新显示
50      MSG msg;
51      while(GetMessage(&msg,NULL,0,0))
52      {
53          TranslateMessage(&msg);       //转换键盘消息
54          DispatchMessage(&msg);        //分派消息
55      }
56      return 0;
57  }
58  LRESULT CALLBACK WinSunProc(
59                          HWND hwnd,        /*窗口句柄*/
60                          UINT uMsg,        /*消息*/
61                          WPARAM wParam,    /*参数 1*/
62                          LPARAM lParam     /*参数 2*/
63                          )
64  {
65      char tmsg[128] = {0};
66      int num1, num2, num3, num4,num5;      //声明 5 个变量
67      num1 = 3+8;                           //加法运算
68      num2 = 10-7;                          //减法运算
69      num3 = 100*33;                        //乘法运算
70      num4 = 155/5;                         //除法运算
71      num5 = 9%2;                           //取模运算
72                                            //把运算符和结果输出到 tmsg 中
73      sprintf(tmsg, "3+8=%d 10-7=%d 100*33=%d 155/5=%d 9%%2=%d",
74              num1, num2, num3, num4, num5);
75      if(uMsg == WM_PAINT)                  /*判断是否是更新窗口消息*/ */
76      {
77          HDC hDC;                          /*定义 DC 设备*/
78          PAINTSTRUCT ps;
79          hDC=BeginPaint(hwnd,&ps);         /*得到设备 hDC*/
80          TextOut(hDC,150,0,tmsg,strlen(tmsg));
81          EndPaint(hwnd,&ps);
82      }
83      else if(uMsg == WM_CLOSE)             /*判断是否单击关闭按钮*/
84      {
85          if(IDYES==MessageBox(hwnd,"是否真的结束？","游戏开发",MB_YESNO))
86          {                                 /*if 语句嵌套*/
87              DestroyWindow(hwnd);          /*单击“确定”按钮，销毁窗口*/
88          }
89      }
90      else if(uMsg == WM_DESTROY)           /*判断是否是销毁窗口消息*/
91      {
92          PostQuitMessage(0);               /*退出程序*/
93      }
94      else
95      {                                     //调用 DefWindowProc
96          return DefWindowProc(hwnd,uMsg,wParam,lParam);
```

```
97      }
98      return 0;
99  }
```

这段代码最后的输出结果，如图 3.3 所示。虽然同样达到了效果，但却没有采用 switch 语句时的阅读性高。现在再回去看看示例代码 3.1，是不是阅读起来比较容易理解。采用 switch 语句时的执行流程如下所述。

（1）计算 switch 后面括号内的表达式的值。

（2）用表达式的值与常量表达式 1 的值进行比较，如果不相等，再与常量表达式 2 的值进行比较，如果还不相等，则顺序比较下去，直到常量表达式 n，如果还是不相等，则执行 default 子句中的语句，如果没有 default 子句，则执行 switch 语句后的其他语句。

（3）在前面比较过程中如果有一个 case 子句中的常量表达式与条件表达式的值相等，则执行该 case 子句中的语句。执行完成后，如果没有遇到 break 语句（将在后面的章节进行讲解），则执行下一个 case 子句中的语句，直到遇到 break 语句退出 switch。

（4）如果后面的语句中都没有 break 语句，则依次执行后续语句。直到 switch 语句的右括号“}”，退出 switch 语句，执行其他语句。

3.6.3　循环语句

计算机能够受程序的控制进行周而复始的重复性工作，就是程序中的循环语句在起作用。循环语句相当于操场上的长跑，当围着操场跑完一圈（执行语句一次），如果没有达到规定的路程，那么还将重新跑一圈（再次执行相同语句一次），直到到达规定的路程才能结束（即循环结束）。

在 C++中，循环语句分为 3 种：

- for 循环语句。
- while 循环语句。
- do-while 循环语句。

这些循环语句各有其独自的特点，根据不同的需要进行选择。其共同的特点是根据循环条件来判断是否执行循环体中的语句。在许多情况下，也可以互相替代。

1. while 循环语句

while 循环是结构最简单的循环语句，用关键字 while 来声明。while 单词的英文意思就是“当…就…”，所以也称 while 循环为当型循环。其声明格式如下：

```
while （条件） 语句                    //单一语句
while （条件）{                        //复合语句
    语句 1;
    语句 2;
}
```

意思是“当”条件为“真”时，就执行“语句”。“条件”可以是各种类型的表达式，通过计算该表达式的值，来决定是否执行“语句”。而“语句”就是循环的循环体，可以是一条语句，也可以是复合语句。

当型循环的执行流程为：先计算出“条件”中的表达式的值，如果值为真（非 0），则执行循环体，即语句；如果值为假，则退出循环并执行循环后面的语句。执行完一次循环体后，再次计算“条件”中的表达式的值，如果其值仍为真（非 0），则再次执行循环体，直到“条件”中的表达式的值为假（0），退出循环执行后面的语句。while 循环语句执行流程如图 3.7 所示。

代码 3.4 是一个计算 1～100 中所有数之和的循环语句示例。

代码 3.4　计算 1~100 中所有数之和（使用 while 语句）

```
01      int n = 1, sum=0;
02      while(n<101)                        //判断 n 是否<101
03      {                                   //循环体开始
04          sum+=n;                         //累加 n 值到 sum
05          n++;                            //n++，自增加 1
06      }                                   //循环体结束
```

整段程序代码的执行流程如下：

（1）比较 n 值和 101，看是否小于 101。如果小于，就继续执行；如果大于则结束。

（2）把 sum 的值加上 n 的值，重新赋值给 sum。

（3）n 值每循环一次增加 1。

（4）重复过程（1）～（3）。当循环了 100 次后 n 值就变成 101，不小于 101，于是跳出循环。

最后的程序输出 sum 的值 5050。

2．do-while 循环语句

do-while 循环是 while 循环语句的一个变种。虽然功能和 while 是一样的，但从程序的执行流程上看还是有一定的差别。其声明格式如下：

```
do
    语句
while (条件);
```

其中，do 和 while 是关键字。其和 while 语句不同的是，do-while 是属于先斩后奏型的，是先执行一次“语句”即循环体，然后计算“条件”中的表达式的值。如果其值为真（非 0），再次执行“语句”，如果其值为假（0），退出循环，执行循环后面的语句。do-while 循环语句执行流程如图 3.8 所示。

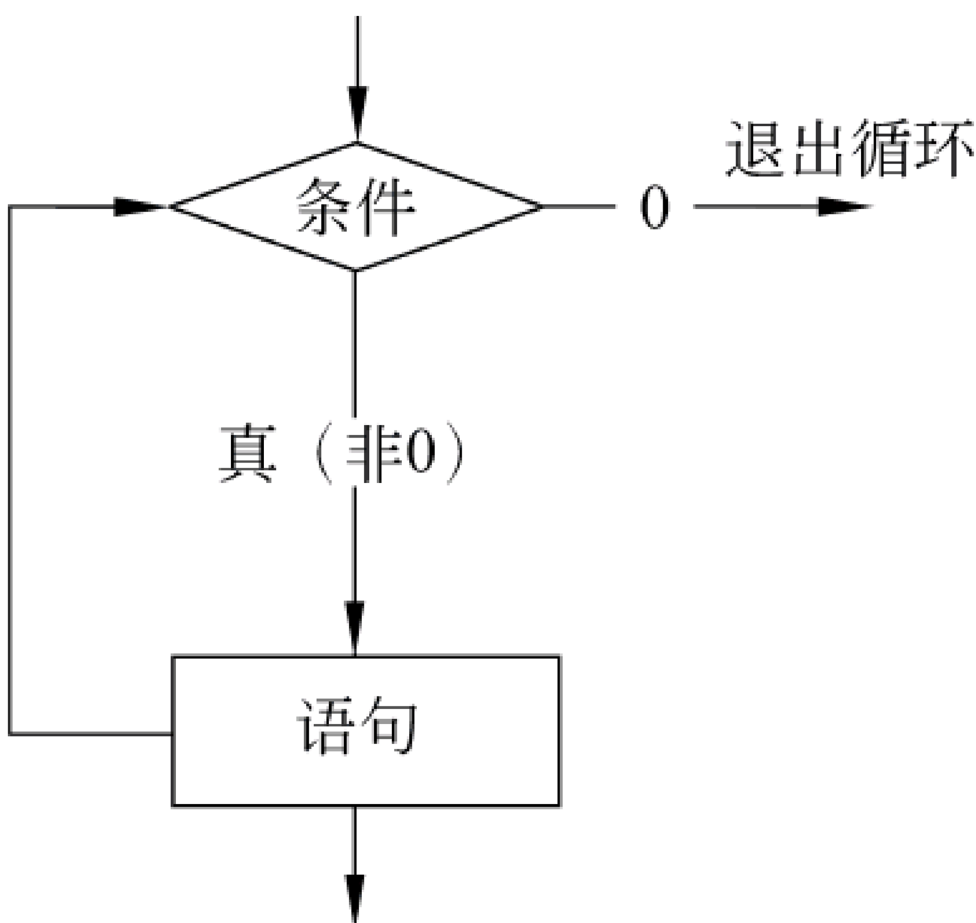


图 3.7　while 循环语句执行流程

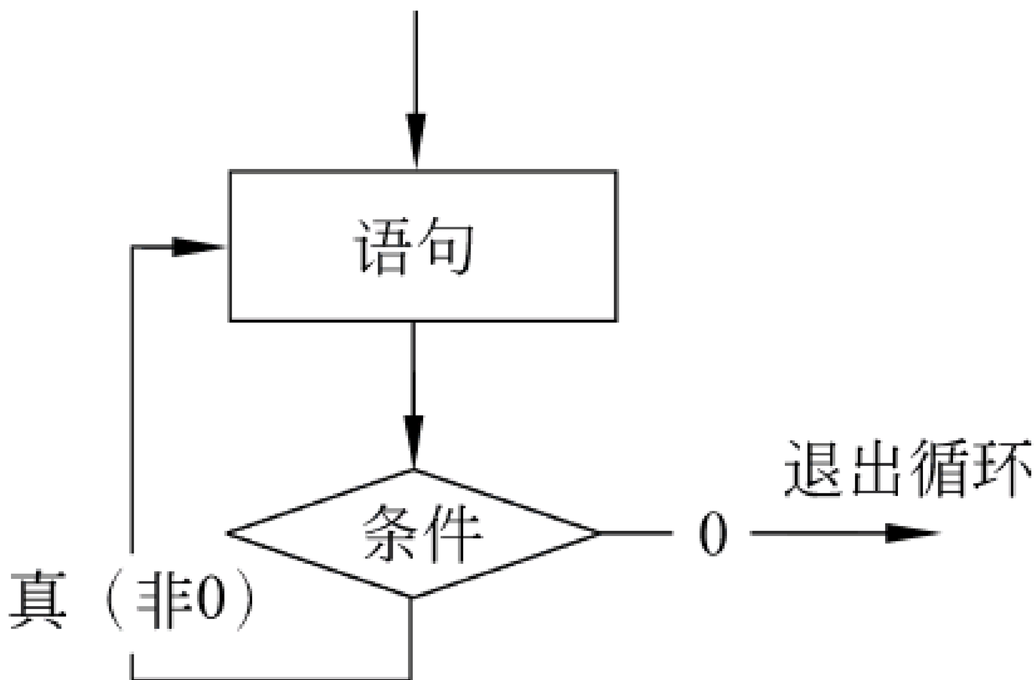


图 3.8　do-while 循环语句执行流程

do-while 和 while 循环语句的区别仅在于 do-while 循环至少执行一次循环体，而 while 循环可以一次都不执行循环体。而在 do-while 中，最后是一个分号结束。代码 3.5 同样是计算 1～100 中的所有数之和，不过采用的是 do-while 循环语句。

代码 3.5　计算 1~100 中的所有数之和（使用 do-while 语句）

```
01        int n = 1, sum=0;
02        do{                                   //先执行语句，循环体开始
03            sum+=n;                           //累加 n 值到 sum
04            n++;                              // n++, 自增加 1
05        }while(n<101);                        //判断 n 是否小于 101，循环体结束
```

整段程序代码的执行流程如下：

（1）把 sum 的值加上 n 的值，重新赋值给 sum。

（2）n 值每循环一次增加 1。

（3）比较 n 值和 101，看是否小于 101。如果小于，就继续执行；如果大于则结束。

（4）重复过程（1）～（3）。当循环了 100 次后，n 值就变成 101，不小于 101，于是跳出循环。

最后输出 sum 的值，仍然是 5050。

3. for 循环语句

循环语句中，除了前面两种外，还有一种循环语句就是 C++中最常用到、形式最多、功能最强的 for 循环语句。其用关键字 for 进行声明。标准的声明格式如下：

```
for （表达式 1;表达式 2;表达式 3） 语句
```

其中，各表达式中间用分号（;）分隔。一般情况下，表达式 1 用来给循环变量初始化，表达式 2 用来判断循环是否结束的条件，如是该表达式为真，则执行循环体；否则退出循环。表达式 3 用来作为循环变量的增加或者减少运算。for 语句的执行流程如下：

（1）计算表达式 1 的值，一般是赋值、初始化或者空表达式。

（2）计算表达式 2 的值，判断是否执行循环体，如果其值为 0，则退出循环，执行循环后的语句，否则执行一次循环体。

（3）计算表达式 3 的值，即改变循环变量的值。

（4）计算表达式 2 的值，并判断是否执行循环体。

for 循环语句执行流程如图 3.9 所示。

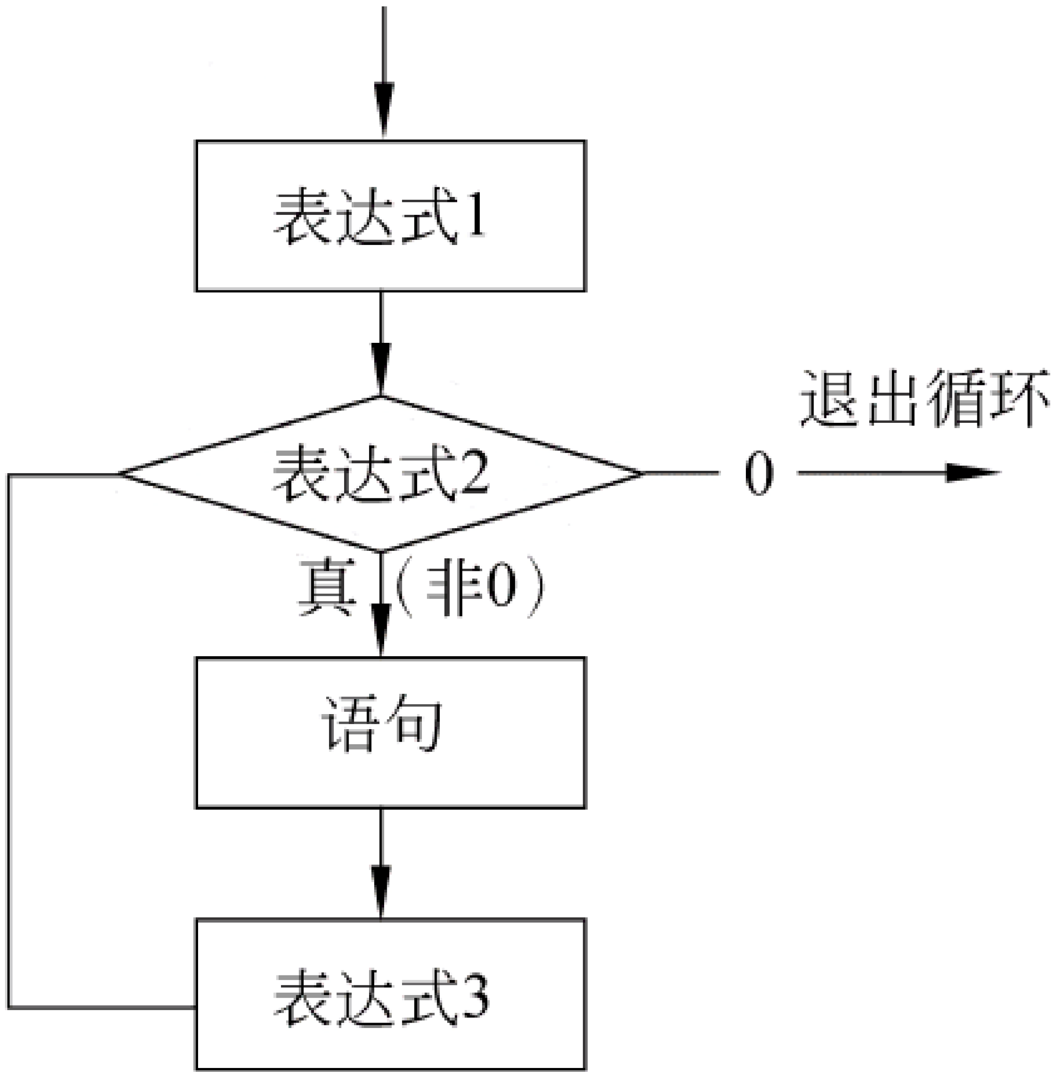


图 3.9　for 循环语句执行流程

其实 for 循环可以用 while 循环来替代，因为 for 循环语句中表达式 1 只计算一次，可以放在 while 循环体外面，而表达式 3 每执行完循环体后计算一次，可以放在循环体中，表达式 2 变成 while 循环的条件，于是 for 循环语句可以变为 while 语句的如下格式：

```
表达式 1;
    while （表达式 2）
{
    语句
```

```
    表达式 3;
}
```

下面用 for 循环来计算 1～100 中的所有数之和，其示例如代码 3.6 所示。

代码 3.6　计算 1~100 中的所有数之和（使用 for 语句）

```
01      int n = 0, sum=0;
02      for(n=1;n<101;n++)             //for 循环语句
03      {                              //循环体开始
04          sum+=n;
05      }                              //循环体结束
```

这段代码就是一个 for 语句，其执行流程如下：

（1）先把 n 值初始化为 1，然后判断 n 是否小于 101，如果是假，退出循环；否则进入循环。

（2）执行 sum+=n。

（3）执行 n++后，把 n 的值增加 1。

（4）继续执行（1）～（3），直到 n 等于 101，退出循环。

代码执行后，最后输出 sum 的值，仍然是 5050。

C++规定，for 语句括号中的 3 个表达式可以写在其他位置，但是两个分号不可省略。请读者自己去实践，这里不再复述。

3.7　C++中的数组、指针及引用

在本节中，读者将学习到 C++中的数组和指针的定义、初始化及操作方法。

3.7.1　数组的定义与操作

在程序设计中，有时候要用到很多的变量来存放数据，但如果变量多了，就会变得难以管理。为此 C++中引入了数组。所谓数组，就是把相同类型的变量编个号放在一组里，这个组就被称为数组。其有 3 个特点：

- 数组是数据的集合，每个数据项都是有序序列的一部分。
- 数组中每个数据项叫做数组的元素。
- 数组是由合法的数据类型构成的线性数据序列。

1．数组的定义

同变量一样，数组也必须先被声明或者定义后才能使用。典型的数组声明的格式如下：

```
数据类型　数组名[大小];
```

其中，数据类型可以是任何一种合法的数据类型，如 int、char、short、long、float、double 等，也可以是后面章节将提到的“类”这种类型。数组名是一个由有效的 C++标识符构成，而方括号[ ]表明这是一个数组，方括号中的数值指定整个数组的大小，即可以存

储多少个元素。现在来看一看如何在 C++中声明和定义一个整型和浮点型数组。

```
int nArray[100];                       //声明一个整型数组
float fArray[10];                      //声明一个浮点型数组
```

在定义一个数组的时候，中括号[]中的大小必须是一个常量数值，因为数组是内存中一块有固定大小的静态空间，编译器必须在编译所有相关指令之前能够确定要给该数组分配多少内存空间。

2．数组的初始化

当声明一个数组时，除非特别指定，否则数组将不会被初始化，因此其内容在将数值存储进去之前是不定的。数组初始化的方法有如下 3 种。

（1）指定大小及初始值的初始化方法，如代码 3.7 所示。

代码 3.7　指定大小及初始值的数组初始化方法

```
01  void func()                          //函数名
02  {
03      int nArray[4]={18,20,30,45};     //数组初始化
04      ...
05  }
```

在上面的例子中，数组 nArray 声明中的长度为 4，因此在后面花括号中的初始值也有 4 个，每个元素对应一个数值。这里需要注意，花括号中要初始化的元素数值个数必须和数组声明时方括号[ ]中指定的数组长度相符。元素之间要用逗号分开。

如果要把数组中 4 个元素全部初始化为 0，初始化方法如下：

```
int nArray[4] = {0};
```

（2）省略数组大小的说明，直接初始化数组，如代码 3.8 所示。

代码 3.8　省略数组大小的数组初始化方法

```
01  void func()                          //函数名
02  {
03      int nArray[]={18,20,30,45};      //数组初始化
04      ...
05  }
```

在上面的例子中，数组 nArray 声明中的大小未指定，而数组的长度将由后面花括号{}中数值的个数来决定。只有当数组立即初始化时才能这样做。

（3）既指定数组大小，又同时对其前几个元素进行初始化的方法，如代码 3.9 所示。

代码 3.9　只对数组前几个元素初始化方法

```
01  void func()                          //函数名
02  {
03      int nArray[10]={1,2,3};          //数组初始化
04      ...
05  }
```

上面例子中就是既指定了数据 nArray 的大小为 10 个元素，同时又把其前 3 个元素分

别初始化为 1、2、3，数据的其他元素被初始化为 0。

3. 数组元素的使用

在 C++程序中可以读取和修改数组任一元素的数值，其操作方法就像其他普通变量一样。格式如下：

```
数组名[元素序列号]
```

数组名加上元素序列号，就可以操作对应数组元素。其中元素序列号又被称为下标。现在回过头来看前面代码 3.8 中定义的数组 nArray，其中包含 4 个元素，每一个元素都是整型 int，要引用其中第 2 个元素的方法如下：

```
nArray[1];
```

读者可能会觉得奇怪，为什么明明引用其中的第 2 个元素，但元素序列号却是 1 呢？这是因为数组元素序列号是从 0 开始索引，不是从 1 开始的。

现在，假设要把数值 20 存入数组 nArray 中的第 3 个元素中，方法如代码 3.10 所示。

代码 3.10　把数值存放在数组中

```
01  void func()
02  {
03      int nArray[] = {18,20,30,45};   //数组初始化
04      nArray[2] = 20;                 //赋值 nArray 内容变成{18,20,20,45};
05      …
06  }
```

而如果要把数组 nArray 的第 3 个元素的值赋给变量 a，或者把这个元素当成一个普通变量使用，其方法如代码 3.11 所示。

代码 3.11　读取数组元素的数值

```
01  main()
02  {
03      int nArray[] = {18,20,30,45};   //数组初始化
04      int a = nArray[2];              //赋值给变量 a，a 的值为 30
05      nArray[2] +=5;                  //元素 3 值为 35
06      …
07  }
```

方括号[ ]在对数组操作中有两种不同的用法：一种是在声明数组的时候定义数组的长度；另一种是在引用具体的数组元素时指明一个索引号（index）。请读者一定不要把这两种用法混淆。

3.7.2　指针的定义与操作

C++语言的强大，就在于其继承了 C 语言中的内存地址操作的能力，而内存地址的操作方式就是通过指针。指针不仅可以获得数据的地址，而且还可以操纵地址。所以指针是 C++中最灵活，也是最难掌握的部分。

1. 指针的定义

指针是一种数据类型，具有指针类型的变量称为指针变量。指针变量是用来存放某个变量地址值的一种变量，和一般变量是不一样的，一般变量存放的是数值，而指针变量中的数据值是某个变量的内存中的地址值。指针本身的类型都是 unsigned long 型的。

一个变量的地址称为该变量的“指针”。有一种变量专门用来存放另一个变量的地址（即指针），则称其为“指针变量”。指针变量的数据类型是由其所指向的变量类型来决定，而不是指针本身数据值的类型决定。这与一般变量的数据类型不同。

由于指针的类型是由其所指向的变量类型决定的，而所指向的变量类型又各不相同，所以指针的类型也是各不相同的。定义指针的方法如下：

```
数据类型 * 指针名;
```

其中，数据类型应该是指针所指向的数据相符合的数据类型。如 int、char、float 等。*表示所声明的是一个指针变量，而不是普通变量。如果这个*号被忽略，就变成声明变量了，笔者当年学习 C++时，就常常犯这个错误，所以初学者一定要多多注意。

指针的变量名应遵循 C++标识符的命名规则。如果要声明多个指针变量时，必须在每个指针变量名前加上*，如下所示。

```
数据类型 * 指针名 1,指针名 2,…;
```

下面列举几种常用的指针变量的声明。

```
int      * pn, *pi;           //pn 和 pi 是两个指向 int 型变量的指针
float    * pl;                //pl 是一个指向 float 型变量的指针
char     * pc;                //pc 是一个指向 char 型变量的指针
int      *(pf)();             //pf 是一个指向函数的指针，该函数的返回值为 int 型数值
int      * *pp;               //pp 是一个指向指针的指针，即二维指针
```

2. 指针的初始化

一旦声明了变量，则放置该变量的地方就在内存中有了地址，可以用&取地址操作符来获取变量的地址。对于一般的变量、数组中的元素等地址值都可以用在变量名前加上&来取得，其格式如下：

```
&变量名;
```

通过这种方式就能够取得变量所在的地址值，例如：

```
int a = 10;
int b[5] = {1,2,3,4,5};
```

取变量 a 的地址值用&a；取数组元素 b[3]的地址值用&b[3]。有了地址值就可以赋值给指针变量对其进行初始化。例如：

```
int *pa = &a;                 //将 a 的地址赋给存放地址的变量（指针）
int *pb = &b[3];              //将 b[3]的地址赋给存放地址的变量（指针）
```

这时，pa 中保存的是 a 的地址值，pb 中保存的是数组元素 b[3]的地址。也可以说 pa

指向了变量 a，pb 指向了数组元素 b[3]。现在假定变量 a 的地址为 8000，数组元素 b[3]的地址为 9006，那么指针和变量的关系如图 3.10 所示。

由于指针变量中只能保存地址，所以不要将任何其他非地址值赋给一个指针变量，例如下面的赋初始值就是不合法的。

```
int *p = 100;
```

3．指针的使用

既然指针是一种数据类型，其也有对应的操作和运算方法。但又由于指针的特殊性，所以指针只有 3 种运算方法。

1）用指针变量加或者减一个整数

在 C++中规定，一个指针变量加或者减一个整数，并不是简单地加或者减一个整数，而是将指针变量的地址和其指向的变量所占用的内存字节数相加或者减。即 p+c*i 代表地址计算。其中，c 为系数（字节数）。整数 c=4，实型数 c=4，只有这样才能保证*（p+i）指向 p 下面的第 i 个元素，才有真实意义。代码如代码 3.12 所示。

代码 3.12　指针的加减法运算

```
01  #include <iostream.h>
02
03  void main()
04  {
05      int a[] = {10,20};                //初始化数组 a
06      float b[]   = {105.5, 103.5f};    //初始化数组 b
07      int * pa    = a;                  //指向数组 a 首地址
08      float * pb  = b;                  //指向数组 b 首地址
09
10      for(int i=0; i<2; i++)
11      {
12          cout<<"pa 中的地址为: "<<pa+i<<"\t 其数值为: "<<*pa+i<<endl;
13          cout<<"pb 中的地址为: "<<pb+i<<"\t 其数值为: "<<*pb+i<<endl;
14      }
15
16  }
```

代码解析：代码第 7 行把 a[0]的地址赋值给指针变量 pa。代码第 8 行把 b[0]的地址赋值给指针变量 pb。而代码第 11～14 行，循环把两个数组中的元素地址赋值给不同的指针，然后输出相应的元素的地址和数值。代码执行结果如图 3.11 所示。

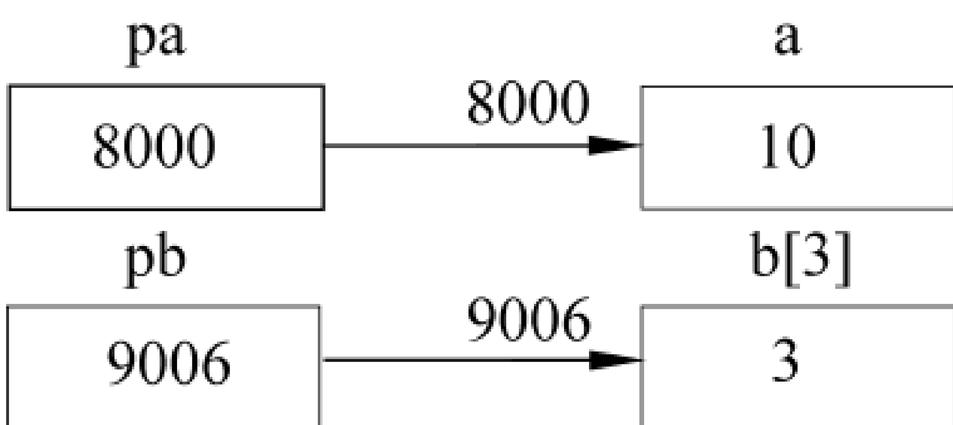


图 3.10　指针与变量的示例

```
pa中的地址为: 0x0012FF78    其数值为: 10
pb中的地址为: 0x0012FF70    其数值为: 105.5
pa中的地址为: 0x0012FF7C    其数值为: 11
pb中的地址为: 0x0012FF74    其数值为: 106.5
Press any key to continue
```

图 3.11　指针的加减法运算输出结果

从执行结果可以看到，其实每次循环中，随着 i 值的增加，指针 pa 和 pb 指向的地址值也相应加了 4。所以把指针和变量 i 值（1 和 2）相加，并不是简单地将指针所指向的地址值同变量 i 相加，而是将指针移动了 4 个字节。

而且还可以从输出结果中看到，指针每增加 1，其指向的变量的数值是数组中下一个元素的数值。数组在内存中是一段连续的存储空间。所以现在得到一个结论，指针同整数的加减法，其实是把指针向前或者向后移动整数个对应类型的存储单元。

2）指针变量赋值

将一个变量或者数组的地址赋值给一个指针变量。例如：

```
int a, array[5];
int p = NULL;
p = array;
p =&a;
```

但要注意，不能把一个整数直接赋值给指针变量，只能将变量已经分配的地址赋值给指针变量。因为这样做会导致访问非法，如图 3.12 所示。

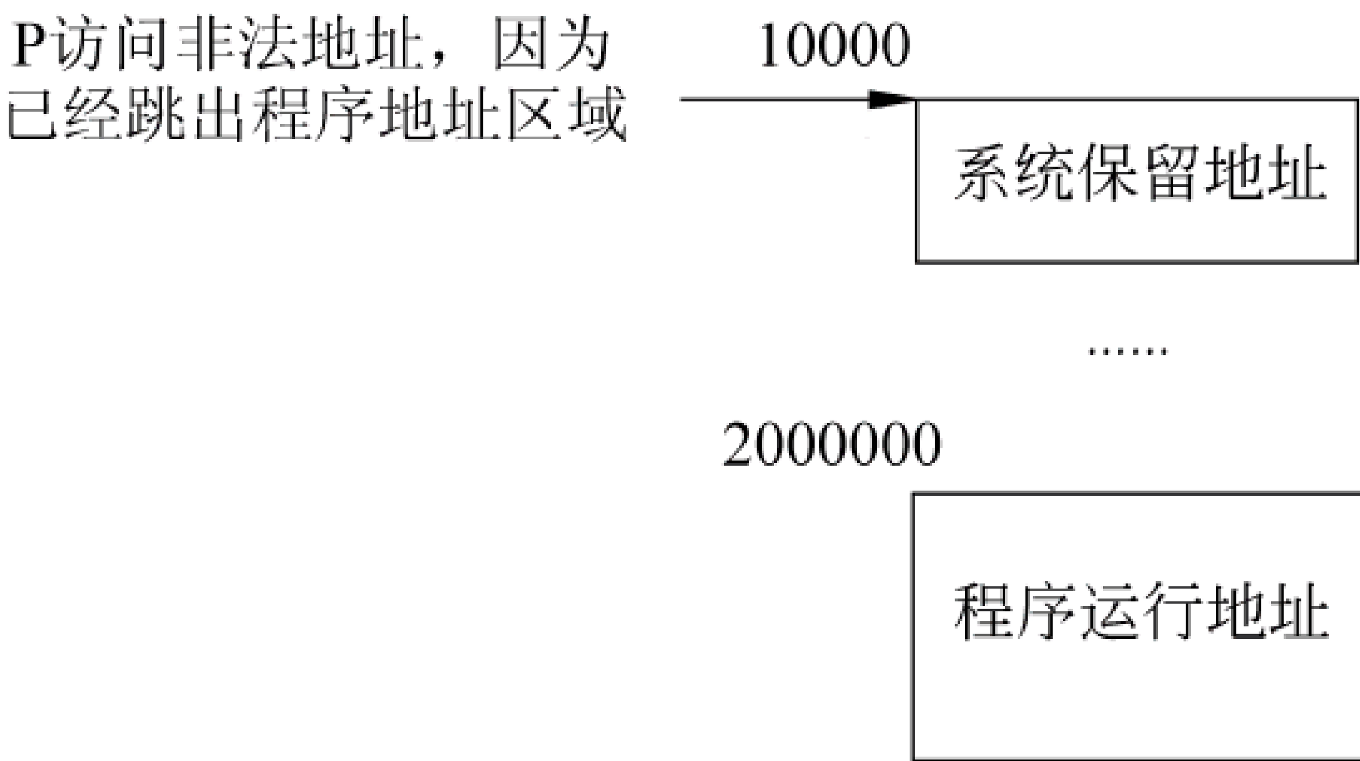


图 3.12　访问非法地址

同样也不应该把一个指针变量赋值给整数变量。例如：

```
p = 10000;        //整数变量赋值给指针变量
a = p;            //指针变量赋值给整数变量
```

3）指针的关系运算

在一定条件下，两个指针可以进行关系运算，例如：两个变量指向同一个数组中的元素，如果将两个指针进行比较，则前面的元素的指针变量“小于”指向后面元素的指针变量。当两个指针相等时，说明这两个指针指向同一元素。因为其涉及的内容较多，所以请读者参考其他相关书籍。

3.7.3　引用的定义与操作

在 C++中还有一种专门存放其他变量“绰号”的变量，叫做引用变量。引用变量通常被认为是另一种变量的别名。

1. 引用的定义

引用定义的格式如下：

```
数据类型 &引用名 变量名;
```

引用名也应遵循 C++标识符的命名规则。例如下面这些就是合法的引用。

```
int      &pn ;             //pn 是对 int 型变量的引用
```

```
float    &pl;             //pl 是对 float 型变量的引用
char     &pc;             //pc 是对 char 型变量的引用
```

2．引用的初始化

一般情况下，定义引用时必须初始化。采用直接初始化方式进行，其格式如下：

```
数据类型 &引用名=变量名;
```

例如：

```
int i = 10;
int &n = i;              //直接初始化
```

这里，n 是一个引用，其是变量 i 的别名。所有在引用上进行的操作，实质上都会作用在被引用者上。例如：

```
n=20;
```

因为 n 是 a 的引用，所以代码的实质是使 i 变为 20。当然也可以将引用赋给另外一个变量，则该变量将具有被引用的变量的值。例如：

```
int m=n;
```

这时，m 具有被 n 引用的变量 i 的值，即 20。引用的用途主要是用来作为函数的参数或者函数的返回值，代码 3.13 是针对引用的示例。

代码 3.13　引用的示例

```
01  void func(int &num)
02  {
03       int &rNum = num;                //对引用 rNum 的初始化
04       rNum = 20;                      //实际是改变 num 的值
05       int temp = rNum;                //把 num 的值赋值给 temp
06  }
```

代码的第 3 行是声明并初始化一个引用，这时引用的就代表变量 num，而代码第 4 行是一个赋值操作，其改变了变量 num 的值。代码最后执行结果为“num 的值变成 20”。

在 C++中使用引用时有如下几点需要注意：

- 引用在定义时要初始化。
- 对引用的操作就是对被引用变量的操作。
- 可以用一个引用给一个变量赋值，该变量的值是被引用的变量值。

3.8　函　　数

学习到这一节，相信读者朋友们已经可以编写简单的程序了。但较大的程序一般由很多个程序模块组成，每一个模块用来实现一个特定的功能，这种程序模块被称为子程序，C++中就是用函数来实现子程序的。

现在来认识什么是函数？所谓“函数”，是由能完成特定功能的独立程序代码块组成

的，可与其他函数结合起来产生最终的输出，函数内部的实现对程序的其他部分是不可见的子程序。

3.8.1 使用函数的好处

使用函数的优点是：函数封闭了一些程序代码和数据，实现了更高级的抽象。在 C++编程中，常常把程序分成多个函数来实现。这样做不仅可以封装或隐藏具体实现的细节问题，实现更高级的抽象，让使用者的精力集中在函数的接口外，而且还实现了参数化和结构化。因此函数抽象的实现，将有利于数据共享，节省开发时间，增强程序的可靠性和便于管理等。

3.8.2 函数的定义及声明

函数的一般定义格式如下：

```
数据类型 函数名(参数表)
{
    语句表达式;
}
```

其中，数据类型是规定了函数的类型，即该函数的返回值（将在下一小节讲解）的类型。其可以是各种合法的数据类型，包含基本数据类型、自定义数据类型、指针和引用类型。函数名即函数的名字，是一种标识符，所以也必须遵循 C++中标识符的规定。一般函数的命名最好做到“见其名便知意”。

例如，代码 3.14 就是一个打印星号*函数的定义。

代码 3.14 打印星号的定义

```
01  int printStar ()                       //使用标识符 printStar 来作为函数名
02  {
03      for(int i=0; i<50; i++)
04      {
05          cout<<"*";
06      }
07      cout<<endl;
08  }
```

代码解析：代码第 2～8 行用花括号{}包含起来的区域便是函数体，是用于打印“*”号。函数名后面紧跟圆括号()用来标识这是一个函数，而不是其他的程序语句或者数据，所以是不可省略的。参数表是可以省略的，在这里读者只要知道其放置的位置就可以，在 3.8.3 节中将进行详细讲解。而用花括号{}包含起来的语句表达式可以是一条语句，也可以是复合语句。

当函数体是由 0 条语句表达式组成的时候，这个函数被称为空函数。而空函数并不是没有意义的。虽然调用此函数时，什么工作也不做，但其是为将来扩充功能做准备的。因为在程序设计之初，不可能所有的函数都是已经写好的，所以在需要扩充的地方写上一个

空函数，先占好位置，等以后用编好的函数代替。这样做可以使程序结构清楚，可读性好，扩充也方便，对程序结果影响不大。例如 mySleep()就是一个空函数。

```
void mySleep(){    }
```

在 C++中函数和变量的有些性质是一样的，对函数的声明是用来指明函数的名字，该函数将在以后的程序体中调用（将在 3.8.3 节讲解）和定义。如果一个函数定义在前，调用在后，调用之前可以不必说明。但如果一个函数定义在后，调用在前，那么调用之前就必须先声明。

3.8.3　认识函数的参数

在 3.8.2 节函数的定义中，已经提到参数列表这个概念，在 C++中因函数的参数列表可以省略，所以函数的形式可以分为以下两类。

- 无参函数：如前面的函数 printStar()就是无参函数。在调用无参函数时，主调函数并不将数据传送给被调用函数，一般用来执行指定的一组操作。无参函数可以有返回值，也可以没有返回值，但一般以不包含返回值的无参函数居多。如 printStar()就只是打印星号函数，其并没有返回值和参数。
- 有参函数：在调用函数时，在主调函数和被调函数之间有参数传递，主调函数可以把数据传给被调函数使用，被调函数中的数据也可以返回来给主调函数使用。

在一般情况下，调用函数时，主调函数和被调函数之间是有数据传递关系的。这就是前面所说的有参函数。有参函数中的参数分为两种：

- 在定义函数时，函数名后面的圆括号“()”中的变量名称为“形式参数”，简称“形参”。
- 在调用函数时，函数名后面的圆括号“()”中的表达式的值称为“实际参数”，简称“实参”。

在有参函数中，要注意参数的以下几点性质。

- 在定义和声明函数中指定的形参变量，在未被其他函数调用之前，其并不占用内存中的存储空间。只有函数被调用时，才会被分配内存空间。在调用结束后，形参变量所占用的空间也会被释放。
- 实参可以是常量、变量或者表达式，不过其必须要有确定的值。在调用时才能将实参的值赋给形参变量。
- 在定义和声明函数中，必须指定形参的类型。
- 实参和形参的数据类型应该一致，这才是合法的。如果是不同的数据类型，就会发生数据类型的转换，虽然编译时不会出错，但有可能导致数据丢失。
- 函数中的参数列表，可以是由一个或者多个参数组成，多个参数中使用逗号进行分隔，但如果被调函数不需要从主调函数那里获得数据，则被调函数的参数列表应该为“空”，可以用 void 来表示，也可以不用。用 void 来表示没有参数的函数，例如：

```
void min(int x,int y);
void fun(int a, int b, int c, int d);
```

```
void fun1(void);
void fun2();
```

- 实参和形参的个数应该一致。
- 实参对形参变量的数据传递是“值传递”，即单向传递，只由实参传给形参，而不能由形参传回给实参。

3.8.4　函数的调用及返回值

一个函数被定义以后就是为了将来对其进行调用及得到返回值。调用函数是实现函数功能的主要手段。得到函数返回值是调用的函数的主要目的之一。所以如何有效地调用函数和得到返回值是 C++的一个重要内容。

1．函数调用

C++中函数调用的方法分为两种，一种是传值调用；另一种是引用调用。而传值调用中又分为传数值调用和传地址调用。

函数的调用是用一个表达式来表示的。其一般调用格式如下：

```
函数名(实参表);
```

其中，实参表是由 0 个、1 个或者多个实参组成的，多个参数之间用逗号隔开，每一个参数可以是变量，也可以是表达式。但实参的个数是由形参的个数决定的。实参在调用函数时给形参进行初始化，所以实参的个数和类型要和形参的个数和类型一致。实参对形参初始化是按其位置对应进行的，即实参表的第 1 个实参值赋给形参表第 1 个形参，第 2 个实参值赋给第 2 个形参，依此类推。

函数的调用是一种特殊的表达式，函数名后的圆括号可以理解为函数调用运算符。函数调用表达式的值就是后面要讲到的 return 语句的返回值。表达式值的数据类型是函数的数据类型。

2．函数返回值

通过调用一个函数后，主调函数能得到一个确实的值，这就是函数的返回值。要获得函数的返回值，被调函数中必须有 return 语句。返回 return 语句有如下两种格式：

- 包含表达式的返回语句。

```
return 表达式;
```

- 不包含表达式的返回语句。

```
return;
```

一般情况下，前一种格式用于带有返回值的被调用函数中；后一种刚好相反，用在无返回值的函数中。

函数的返回值便是返回语句后面表达式的值。具有表达式的返回语句的实现过程如下所述。

（1）先计算出表达式的值。

（2）如果表达式的值类型与函数的类型不相同时，将表达式的类型自动转换为函数的类型，这种转换是强制性的，可能会出现数据丢失的情况。函数值的类型在定义函数时是指定的。例如：

```
int max(int x, int y);                    //函数值为整型
char getchar();                           //函数值为字符型
float min(float a, float b);              //函数值为单精度实数型
```

（3）将计算出的表达式的值返回给主调函数作为被调函数的值，该值可以赋值给变量，也可以直接使用。

（4）结束被调函数，并执行主调函数后面的语句。

例如，代码 3.15 就是一个函数调用和得到函数返回值的例子。

代码 3.15　函数的调用及得到返回值

```
01  int min(int x, int y);                 //声明一个min()函数,其返回值类型为int
02  void func()
03  {
04      int a=100,b=200,c=0;
05      c=min(a,b);                        //调用min()函数，并得到返回值放入c
06  }
07  int min(int x, int y)                  //定义一个函数，返回值类型为int
08  {
09      return(x<y?x:y);                   //返回小的一个数
10  }
```

代码解析：在代码的第 1 行声明了一个函数返回值为 int 型的函数 min()，并在第 7～10 行定义了这个函数，在第 5 行调用函数 min()，通过运算后，函数 min()把结果返回给主调函数 func()，并存入变量 c 中。函数的返回值是通过函数中的 return 语句获得的。return 语句将被调用函数中的一个确定值带回主调函数中去。其传递关系如图 3.13 所示。

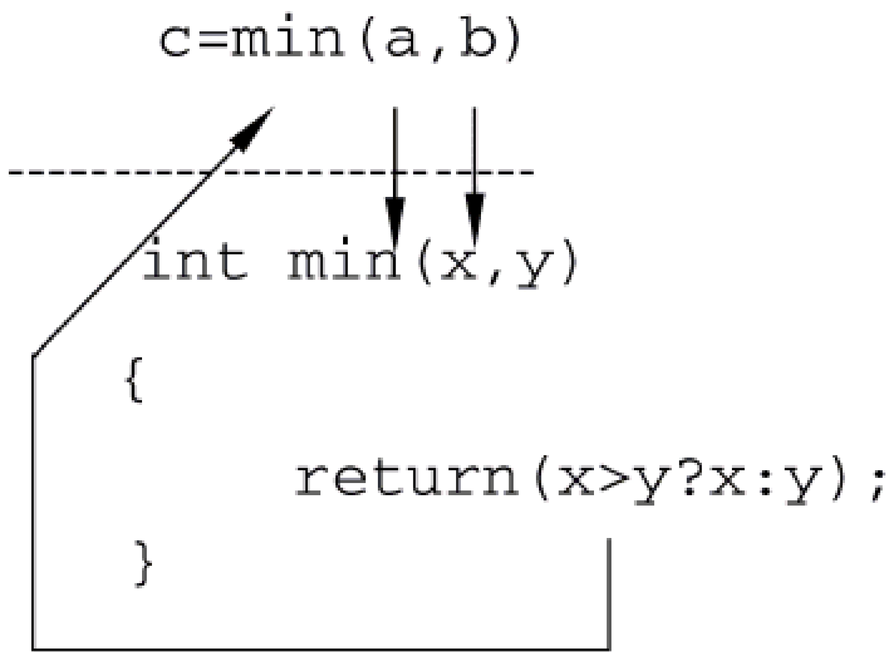


图 3.13　return 语句返回函数值

从图中可以看出，变量 c 最后存放的是表达式(x>y?x:y)的值。即两个数中最小数的值。使用无表达式的返回语句时，只有结束当前的被调函数，才能执行主调函数后面的语句。为了明确表示函数没有返回值，必须用 void 把函数定义为“无类型返回”。

void 类型的函数中可以有无表达式的 return 返回语句，也可以没有 return 语句。当函数中没有 return 语句时，执行到函数体最后一条语句。返回主调函数，相当于函数体的右括号“}”有返回的功能。例如：

```
void min(int x,int y)
{
    ...
    return;
}
void fun(int a, int b, int c, int d)
{ … }
```

函数中可以有多个 return 语句，多数是出现在 if 语句中，当程序执行到这里时，就结束被调函数，并返回主调函数。

3.9　C++的类及其主要函数

既然是面向对象的设计，那么首先要搞明白什么是对象。现实中人们要进行研究的任何事物都是对象，其组成了整个世界。包括抽象的规则、计划或者事件。

对象是一个类的实例，具有自身的状态（一个对象用数据值来描述它的状态）和操作（用于改变对象的状态、对象及其操作）。对象实现了状态和操作的结合，使状态和操作封装于对象的统一体中。这也是对象的特征。

简单地说，就是一些个体真实反映于现实世界中的事物。例如，你、我、他都是对象，是人这个类的实例。

类是具有相同属性和行为的一组对象的集合。它为属于该类的所有对象提供了统一的抽象描述，其内部包括属性（是对象的状态的抽象，用数据结构来描述）和行为（对象操作的抽象，用操作名和实现该操作的方法来描述）两个主要部分。对象的抽象就是类。

类是面向对象程序设计的核心，是实现抽象类型的基础。类是对某一类对象的抽象；而对象是某一种类的实例，因此类和对象是密切相关的。没有脱离对象的类，也没有不依赖于类的对象。

C++中的类是一种复杂的、用户定义的数据类型，它是将不同类型的数据和与这些数据相关的操作封装在一起的集合体。正因为 C++中提供了类这种工具，使得实际生活中应用的各种对象，在程序中可以直接被表示为一个标识符，并可以对其进行引用和操作。属性由类中的数据成员来表示，而操作由类中的成员函数来表示。

3.9.1　C++的优点

与 C 语言相比，C++主要的优点表现在类上。类概念的引入能帮助程序员更好地描述由对象个体组成的现实世界。类有 3 个重要的性质：封装性、继承性、多态性。这些性质在软件的可重用性、可扩充性以及设计的维护方面，具有重要的作用。

1．封装性

把数据和方法有机地联系在一起形成一个具有类特征的对象。封装好的对象就具有明确的功能和方便的接口，以便其他类引用。另外，封装的对象也有各种访问权限，防止被外界非法获取或者更改。

2．继承性

继承性是面向对象程序设计中最重要的机制之一。其改变了过去面向过程的程序设计中，那种编写出来的程序因为无法适应用户多变的需求，而进行改写甚至重写的方法。大大地提高了程序的重用性，减少了资源的浪费。

面向对象程序设计的继承机制给大家提供了无限重复利用程序资源的途径。通过继承机制，可以扩充和完善旧的程序以适应新的需求，这样不仅节省了程序开发的时间和资源，也为以后的程序增添了新的资源。

同类事物具有共同性，在同类事物中，每个事物又具有特殊性。运用抽象的原则舍弃对象的特殊性，抽取其共同性，则得到一个适应于一批对象的类，这便是基类（父类），而把具有特殊性的类称为派生类（子类），派生类的对象拥有其基类的全部或部分属性与方法，称做派生类对基类的继承。

3. 多态性

所谓多态性是指对同一种消息，多种对象有完全不同的行为表现。例如，同时给几个人送花，每个人的表现都会有所不同，有些人是开心；有些人是疑惑；有些人是惊讶；有些人是无所谓。这里 C++所说的消息主要是成员函数的调用，而不同的行为表现则是指成员函数不同的实现。这样，也与现实中的处理方式相符合。

具体来说，可以用“一个对外接口，多个内在实现方法”来表示。再举一个例子，计算机中的堆栈可以存储各种格式的数据，包括整型、浮点或字符。不管存储的是何种数据，堆栈的算法实现是一样的。针对不同的数据类型，编程人员不必手动选择，只需使用统一接口名，系统可自动选择。

3.9.2　定义 C++类

类的定义格式一般分为声明和实现两个部分。其中声明部分用来说明类中的成员，包含数据成员的声明和成员函数的声明。实现部分是用来对成员函数的实现。换句话说，声明部分是告诉使用者“干什么”，而实现部分是告诉使用者“怎么干”。

一般使用者主要关心的是声明部分，实现部分则不是很关心。一般把类的声明放在头文件中，实现放在源文件中，这样方便使用时引用，也实现了代码隐蔽性。类的声明格式如下：

```
class 类名
{
    成员函数的声明
    数据成员的声明
};
```

其中，class 是定义类的关键字，类名是符合规范的标识符，一般情况下，采用 C 字母开始的字符串作为类名。例如，CXXXX 用来表示这是一个类名，以方便与对象名、函数名等相区别。一对花括号“{}”是类的说明部分，称为类体。类体中不仅包含数据成员和成员函数，同时还要明确其保护级别。

保护级别分为 3 种：public 公共访问、protected 保护访问和 private 私有访问。作用如下：

- ❑ public 定义的成员，就是类对外的接口，外部只能通过这个接口，才能访问类的成员。其一般是成员函数，以方便在程序中调用。
- ❑ protected 和 private 定义的成员，一般是类的数据成员，主要用来描述类的对象的属性，用户无法直接访问和使用。只有类中成员函数才能对其进行访问和使用。

类的访问控制关键字可以在类体内出现多次，而且与出现的先后顺序无关。例如：

```
class CDemo
{
public:
```

```
    …
private:
    …
protected:
    …
public:
    …
}
```

类体中的访问控制关键字的作用域，是从控制关键字开始直到下一个控制关键字出现，如果后面没有其他控制关键字，则在“}”前结束。

为了方便说明，现在举一个 C++的类的例子——CMyClass 类，如代码 3.16 所示。

代码 3.16　CMyClass 的声明

```
01  class CMyClass
02  {                       //定义成员变量
03      public:             //定义公有成员变量
04          bool bflg;
05      private:            //定义私有成员变量
06          int x, y;
07                          //定义成员函数
08      public:             //定义公有成员函数
09          CMyClass();     //构造函数
10          ~CMyClass();    //析构函数
11          int GetX();     //成员函数
12          int GetY();     //成员函数
13      protected:          //保护成员函数
14          void SetPoint(int x, int y);
15
16  };
```

3.9.3　成员变量

在类的定义中规定，类体中声明的变量、指针、数组或者对象都是类的成员，称为数据成员。又因数据成员主要用于保存数据，所以其又被称为属性。数据成员的类型可以是任意的数据类型。包括整型、实型、字符型、布尔型、结构、枚举型等。类的数据成员的声明格式如下：

```
class 类名
{
    数据类型 数据成员名;
};
```

另外，类的数据成员也可以是另一个类的对象。但是要注意，如果是另一个类的对象作为本类的成员时，另一个类的声明应该在本类之前，而且成员变量的初始化应当放在构造函数中进行。

在代码 3.16 中，CMyClass 类的成员变量包括布尔型的 bflg 和整数型的 x、y 这 3 个。其中 bflg 定义为 public 访问，是类对外的接口之一。x 和 y 定义为 private 属性，只有 MyClass 类自身能够访问操作。

由于成员变量在类定义的时候进行定义，因此该类中的所有函数都能有效地访问这些变量。对于 C++初学者而言，要注意成员变量、全局变量和临时变量的区别。

在类的代码实现文件（一般是“Cpp”文件）中，起始位置定义的、不包含在任何函数中的变量被称为该类的全局变量，可以被这个类的所有成员函数访问和操作，类似于类的成员变量。

在类的函数代码实现的时候，包含在某个函数中，临时说明并使用的变量，被称为临时变量。临时变量只能在定义它的函数内部访问和操作，类的其他成员函数无权访问和操作。如果临时变量的名称与某个成员变量名相同，则会导致该成员变量在这个函数中失效，这点在实际开发时要注意避免。

3.9.4　成员函数

在类的定义中规定，类体中声明的函数作为类的成员，称为成员函数。又因为一般成员函数是用来对数据成员进行操作的，所以又被称为“方法”。声明成员函数的格式如下：

```
class 类名
{
    数据类型 成员函数名;
}
```

其中，数据类型是成员函数的数据类型，即返回值。成员函数名，即成员函数的名称。

在代码 3.16 中，CMyClass 类的 GetX()和 GetY()就是成员函数，其访问权限是 public 的，所以其也是类对外的接口之一。

3.9.5　构造函数

在 C++中，就有一种特殊的成员函数，是在创建对象时，用来给对象进行初始化的函数，即设定对象的初始值，被称为构造函数。构造函数的声明格式如下：

```
class 类名
 {
    函数名();
};
```

构造函数有如下几个特点：

- 构造函数的函数名，必须和类名相同。
- 构造函数是没有返回值的特殊成员函数。
- 一个类可以有多个构造函数，各构造函数的参数表不同。
- 构造函数可以不带参数，这种构造函数被称为默认构造函数。
- 在构造函数中，可以给出各成员变量的默认值。

代码 3.17 定义了几种不同的 CMyClass 类的构造函数。

代码 3.17　不同的 CMyClass 构造函数

```
01  class CMyClass
02  {                                           //定义成员变量
```

```
03      public:                               //定义公有成员变量
04          bool bflg;
05      private:                              //定义私有成员变量
06          int x, y;
07                                            //定义成员函数
08      public:                               //定义公有成员函数
09          CMyClass();                       //默认构造函数
10          CMyClass(int x);                  //带一个参数的构造函数
11          CMyClass(int x, int y);           //带两个参数的构造函数
12          CMyClass(int x=0, int y=10);      //参数带默认值的构造函数
13          ~CMyClass();                      //析构函数
14          int GetX();                       //成员函数
15          int GetY();                       //成员函数
16      protected:                            //保护成员函数
17          void SetPoint(int x, int y);
18  };
```

3.9.6　析构函数

析构函数的功能刚好和构造函数相反，主要用来释放一个对象。析构函数是特殊的成员函数，析构函数的名称和类名相同，不过其前面必须加“~”字符，用来与构造函数区别。析构函数没有返回值类型。析构函数的声明格式如下：

```
class 类名
 {
    ~析构函数名();
    …
};
```

在代码 3.17 的 CMyClass 类中，成员函数~CMyClass()就是这个类的析构函数。析构函数的特点如下：

- 析构函数名称必须以~开始，其余部分与类名完全相同。
- 在析构函数声明的时候，不能包含任何返回类型。
- 析构函数不能带参数。
- 一个类只能有一个析构函数。
- 在程序运行时，当类的对象越界的条件下，自动调用类的析构函数。

3.9.7　虚函数

如果某类中的一个成员函数被声明为虚函数，那么其成员函数在子类中就可以有不同的实现。一般虚函数的声明格式如下：

```
virtual 数据类型 函数名(参数表)
```

其中，virtual 是关键字，用于声明虚函数。当使用这个成员函数操作对象指针或者对象引用时，采取动态联编方式，在运行时进行关联或者指定。虚函数的作用是使类更好地支持多态性。

当一个基类中的函数为虚函数，则其派生类中也都是虚函数。代码 3.18 就是虚函数的示例。

代码 3.18　虚函数示例

```
01  #include <iostream.h>
02  class CPoint                          //声明 CPoint 类
03  {
04  public:
05      CPoint(float w, float h);         //构造函数
06      virtual float getArea();          //把求面积函数声明为虚函数
07  private:
08      float m_w, m_h;
09  };

10  CPoint::CPoint(float w, float h)      //带参数的构造函数
11  {
12      m_w = w;
13      m_h = h;
14  }
15  float CPoint::getArea()               //虚函数实现
16  {
17      return 0;
18  }
19
20  class CRect:public CPoint             //声明 CRect 类公有继承于 CPoint 类
21  {
22  public:
23      CRect(float w, float h);
24      virtual float getArea();          //把求面积函数声明为虚函数
25  private:
26      float m_w, m_h;
27  };
28  CRect::CRect(float w,float h)
29  :CPoint(w,h)                          //用成员初始化列表来调用父类的构造函数
30  {
31      m_w = w;
32      m_h = h;
33  }
34  float CRect::getArea()                //虚函数实现
35  {
36      return m_w * m_h;
37  }
38
39  void disp(CPoint &p)                  //普通显示函数的实现
40  {
41      cout<<"面积为 : "<<p.getArea()<<endl;
42  }
43  void main()
44  {
45      CRect rt(10.0,5.0);               //创建 CRect 类的对象 rt
46      disp(rt);
47  }
```

代码第 6 行，声明类 CPoint 的成员函数 getArea()为虚函数。代码第 24 行，声明子类中的成员函数 getArea()为虚函数。因为这个成员函数与父类中的成员函数同名，而且参数和返回类型相同，在类 CPoint 中也已经被声明为虚函数，所以其在子类 CRect 中 virtual

关键字是可以被省略的，而且默认为虚函数。

代码第 39 行，由于类 CPoint 中声明了虚函数，所以 disp()函数的对象引用参数 a 被动态联编，其调用的 getArea()成员函数，会在程序运行时根据其代表的对象是 CRect 类的对象自动调用 CRect 类中的对应函数。代码最后输出结果为：

```
面积为:25
```

从最后的输出结果可以看出，设置了虚函数后，disp()函数就变成负责显示所有对象面积的功能函数，其只要调用求面积的 getArea()函数就行了，不管参数是什么类型的对象，C++的动态联编会做好这一切的。这样一来，disp 写好后就不用修改，程序也显得简单，以后如果要求一个新的形状的面积，只要简单地增加一个类就行了。

3.10　运算符的重载

相信大家对于前面学习的关于基本数据类型的操作运算符应该比较熟悉了，因为在 C++的很多语句中都会使用到。不过对于类的对象，这些操作运算符就需要做一些特殊的声明和实现，才可以变成支持对象运算的多功能操作符。现在请大家先来看看下面的操作是有效的吗？例如：

```
int a = 10, b= 10;
a=a+b;              //加法操作
b=b-a;              //减法操作
a=a*b;              //乘法操作
a=a/b;              //除法操作
a=a%b;              //取模操作
```

其实，上面这些操作都是有效操作，因为操作符两边的变量都是基本数据类型。但如果操作符两边变成了对象，会怎样呢？例如：有一个类 CA，程序中的 a、b、c 都是这个类的对象。现在想要把对象 a 和对象 b 的值求和并赋值给对象 c，如下所示。

```
class CA;
CA a,b,c;
c=a+b;
```

上面的操作会产生错误，因为相加运算符的两边并不是基本数据类型。不过，因为 C++允许对运算符进行重载，所以这个相加操作就不会出错了。

甚至还可以修改这个运算符的效果，但如果在类中没有对这个操作符进行重载，那么还是会产生错误的。在类中对一个操作符进行重载后，操作符的功能就变得多种各样。

在 C++中常用的允许被重载的操作符，如表 3.17 所示。

要想重载一个操作符，在对应的类中需要编写一个成员函数，名为 operator，后面跟想要重载的操作符即可。其格式如下：

```
class 类名
{
    数据类型 operator 操作符(参数表);
};
```

表 3.17　允许重载的操作符

操　作　符	说　　明	操　作　符	说　　明
+	加	%	取余
–	减	&	按位与
*	乘	^	按位非
/	除	!	取反
=	赋值	|	按位或
<	小于	&&	与操作
>	大于	||	或操作
==	等于	new	生成堆空间
!=	不等于	delete	释放堆空间
<=	小于等于	[]	数组元素访问
>=	大于等于		
++	自增		
– –	自减		

其中，数据类型可以是类，也可以是基本数据类型。operator 是关键字，操作符即是重载的操作符。下面举例说明如何重载操作符及使用，如代码 3.19 所示。

代码 3.19　操作符的重载及应用

```
01  #include <iostream.h>
02  #include <string.h>
03  class CString                              //声明 CString 类
04  {
05  public:
06      CString();                             //默认构造函数
07      CString(char * pStr);
08      char * getStr();
09      CString operator+(CString &t);         //重载操作符+
10  private:
11      char str[128];
12  };
13  CString::CString()
14  {
15      str[0]='\0';                           //把字符数组清空
16  }
17  CString::CString(char * pStr)
18  {
19      strcpy(str, pStr);
20  }
21  char * CString::getStr()
22  {
23      return str;
24  }
25  CString CString::operator+(CString &t)        //操作符重载的实现
26  {
27      CString tmp;
28      strcat(str, t.str);                //把当前对象的字符串和参数的字符相加
29      strcat(tmp.str, str);
30      return tmp;                        //返回临时对象
31  }
```

```
32
33  void main()
34  {
35      CString str1("this is "), str2("operator overload!"), str3;
36      str3 = str1+str2;               //调用重载后的操作符“+”，实现字符串相加
37      cout<<str3.getStr()<<endl;
38  }
```

代码解析：代码第 9 行，是对操作符“+”的重载声明，这个声明看上去有点复杂；代码第 15 行，是对数据成员赋初值，相当于清空字符数组，因为字符数组的第一个元素变成了空，计算机会自动认为其只是一个空数组。

代码第 25 行是重载操作符函数的实现，先把当前对象的字符数组和形参引用对象的字符数组进行组合，然后再把组合后的数组和临时对象的数组进行组合，最后返回临时对象。代码第 36 行是两个对象相加，并赋值给另一个对象 str3，因为类 CString 进行了操作符+的重载，所以编译不会产生错误。代码最后输出结果为：

```
this is operator overload!
```

实现了运算符重载之后，对象与对象之间也可以进行加法运算，这样不仅提高了程序的可读性，而且也显得更直观了。

3.11　C++语言的编程规范

随着 C++语言的发展，在程序代码的编写上就形成了一套规范。其目的是为了增强程序的可读性，减少阅读中的各种误解。同时为程序的交流和代码的更新提供很大的方便。无论对于熟练的程序员还是初学者来说，都应当尽量遵守这个规范。但要注意，不符合规范的代码并不表示不能被编译。

3.11.1　命名规范

一段很长的 C++程序必然会拥有很多变量，如果不遵守规范的命名方式，每个变量的含义和作用就不容易记住，导致阅读和编写代码的效率降低。微软公司提供了一套称为“匈牙利命名法”的约定命名方式，为每一个变量增加前缀（如表 3.18 所示），来表示这个变量的特定含义，而且要求变量名本身也要有明确的含义。

表 3.18　匈牙利命名法中各前缀的含义

前　缀	正　常　名	含　义
a	array	数组
b	boolean	布尔型
by	byte	字节型
c	character	字符型
cb	count-byte	字节计数
cr	color	颜色

续表

前　缀	正　常　名	含　义
dw	double-word	双字节长整数
fn	function	函数
h	handle	句柄
i	integer	整数
l	long	长整型
lp	long-pointer	长指针
m_	member	成员变量
np	near-pointer	近指针
p	point	指针
s	string	字符串
sz	string-zero	以‘\0’为结束符的字符串
tm	text-measure	正文大小
u	unsigned	无符号
w	word	字
x		坐标 x
y		坐标 y

对于临时变量，如循环控制中的计数变量 n，不会造成可读性方面的问题。而对于重要的非临时性变量，不要用简单的单个字母作为变量名，尽量用能表示该变量作用的英文单词作为变量名。如字符串可以用 MyString、计时变量可以用 MyTime 等。对每个人来说，当规范的命名方式成为习惯后，彼此之间的程序交流就会变得简单易行。

除了匈牙利命名法中列举的前缀含义外，还要注意下面几个常用的规范。

（1）说明较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。例如，如下单词的缩写能够被大家基本认可。

```
temp 可缩写为  tmp  ;
flag  可缩写为  flg  ;
statistic 可缩写为  stat ;
increment 可缩写为  inc  ;
message 可缩写为  msg  ;
```

（2）命名中若使用特殊约定或缩写，则要有注释说明。应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

（3）自己特有的命名风格，要自始至终保持一致，不可来回变化。即命名规则中没有规定到的地方才可有个人命名风格。

（4）对于变量命名，禁止取单个字符（如 i、j、k……），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但 i、j、k 作局部循环变量是允许的。

（5）除非必要，不要用数字或较奇怪的字符来定义标识符或者变量。例如，下列命名会使人产生疑惑。

```
#define _EXAMPLE_0_TEST_
#define _EXAMPLE_1_TEST_
```

```
void set_sls00( BYTE sls );
```

应改为有意义的单词命名：

```
#define _EXAMPLE_UNIT_TEST_
#define _EXAMPLE_ASSERT_TEST_
void set_udt_msg_sls( BYTE sls );
```

（6）用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。表 3.19 是一些在软件中常用的反义词组。

表 3.19　常用的反义词组

add/remove	begin / end	create / destroy
insert / delete	first / last	get / release
increment / decrement	put / get	add / delete
lock / unlock	open / close	min / max
old / new	start / stop	next / previous
source / target	show / hide	send / receive
source / destination	cut / paste	up / down

用有互斥意义的词组来命名的变量或相反动作的函数示例：

```
int  min_sum;
int  max_sum;
int  add_user( BYTE *user_name );
int  delete_user( BYTE *user_name );
```

（7）除了编译开关/头文件等特殊应用，应避免使用_EXAMPLE_TEST_之类以下划线开始和结尾的定义。

3.11.2　格式规范

在 Visual C++中有很多的固定格式，读者在使用其编写代码时也就尽量遵守，例如：

- 尽量使用 Tab 键盘对齐代码，Tab 键的默认值是 4 个字符。
- 使用程序块的分界符花括号{}时，应各独占一行并且位于同一列，同时与引用它的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义，以及 if、for、do、while、switch、case 语句中的程序都要采用左对齐的方式。由于一般情况下，两个花括号之间有时会相隔很多行代码，一定要注意匹配。如果不注意，也容易造成结构的混乱。

如下面的例子就是不符合规范的。

```
for (...) {
    ... // program code
}
if (...)
    {
    ... // program code
    }
void example_fun( void )
    {
```

```
    ... // program code
    }
```

按照规范应采用如下代码格式进行书写。

```
for (...)
{
    ... // program code
}
if (...)
{
    ... // program code
}
void example_fun( void )
{
    ... // program code
}
```

- 同一行不要写多条语句，同一行也不要写不同类型的语句。如下面的书写格式就不符合规范。

```
void func()
{
    int a = 1; float b=0.3F
    if(a>0) b=0.5;
    test.init();test.add(a);
}
```

正确的书写方式如下：

```
void func()
{
    int a= 1;
    float b=0.3F;
    if(a>0)
    {
        b=0.5;
    }
    test.init();
    test.add(a);
}
```

- 在必要的位置加入空格，如赋值号“=”的两边，使代码看上去不堆挤在一起，直接影响可读性。

3.11.3　函数规范

在函数编写和调用时，也有一些小细节应该注意，下面列举几点。

（1）对所调用函数的错误返回码要仔细、全面地处理。

（2）明确函数功能，精确地实现函数设计，而不是近似的功能。

（3）编写可重入函数时，应注意局部变量的使用，应使用 auto 即默认状态的局部变量或寄存器变量。不应使用 static 局部变量，否则必须经过特殊处理，才能使函数具有可重入性。

（4）编写可重入函数时，若使用全局变量，则应通过关中断、信号量（即 P、V 操作）

等手段对其加以保护。若对所使用的全局变量不加以保护，则此函数就不具有可重入性，即当多个进程调用此函数时，很有可能使有关全局变量变为不可知状态。

（5）应明确规定对接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责，默认是由函数调用者负责。对于模块间接口函数的参数合法性检查这一问题，往往有两个极端现象，即要么是调用者和被调用者对参数均不作合法性检查，结果就遗漏了合法性检查这一必要的处理过程，造成问题隐患；要么就是调用者和被调用者均对参数进行合法性检查，这种情况虽不会造成问题，但产生了冗余代码，降低了效率。

（6）防止将函数的参数作为工作变量。因为这样有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。例如下面的函数实现就不太好。

```
void sum_data( unsigned int num, int *data, int *sum )
{
    unsigned int count;
    *sum = 0;
    for (count = 0; count < num; count++)
    {
        *sum  += data[count]; // sum 成了工作变量，不太好
    }
}
```

若改为如下，则更好些。

```
void sum_data( unsigned int num, int *data, int *sum )
{
    unsigned int count ;
    int sum_temp;
    sum_temp = 0;
    for (count = 0; count < num; count ++)
    {
        sum_temp += data[count];
    }
    *sum = sum_temp;
}
```

（7）函数的规模尽量限制在 300 行以内。不包括注释和空格行。

（8）一个函数仅完成一个功能。

（9）为简单功能编写函数。虽然为仅用一两行就可完成的功能去编函数好像没有必要，但用函数可使功能明确化，增加程序的可读性，方便维护、测试。

示例：如下语句的功能不是很明显。

```
value = ( a > b ) ? a : b ;
```

改为如下就很清晰了：

```
int max (int a, int b)
{
    return ((a > b) ? a : b);
}
value = max (a, b);
```

（10）不要设计多用途面面俱到的函数。多功能集于一身的函数，很可能使函数的理

解、测试、维护等变得困难。

（11）函数的功能应该是可以预测的，也就是只要输入数据相同就应产生同样的输出。带有内部“存储器”的函数的功能可能是不可预测的，因为其输出可能取决于内部存储器（如某标记）的状态。这样的函数既不易于理解又不利于测试和维护。

3.11.4　其他规范

除了前面的规范外，还有一些其他的规范需要注意，如下所示。

（1）避免在一行代码或表达式的中间插入注释。除特殊情况外，不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

（2）通过对函数或过程、变量、结构等正确的命名，以及合理地组织代码的结构，使代码一看就明白其意义。清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

（3）在代码的功能、意图层次上进行注释，提供有用、额外的信息。注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

（4）避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

（5）编程时要经常注意代码的效率。代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

（6）在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。但不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

（7）局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。

（8）通过对系统数据结构的划分与组织的改进，以及对程序算法的优化来提高空间效率。这种方式是解决软件空间效率的根本办法。

3.12　总　　结

在这一章中，读者已经学习到很多关于 C++编程语言的基础知识，包括 C++编程语言的优点及其发展过程、数据类型、各种控制结构及类知识。此外，在本章中出现的这些基本概念都是本书后面章节中将要学习的基础，足见其重要性。

总之，掌握了这些基本知识后，就可以迅速地开发 C++应用程序，来解决一些基本问题。掌握好这些内容，就可以使你能够真正成为一名游戏设计者。在第 4 章读者将学习到网络的相关知识如下：

- TCP/IP 协议的构成。
- Socket 的概念及其网络通信模式。
- Windows Sockets 类的介绍及使用。

3.13　挑　　战

一、问答题

1．在 C++中，运算符分为哪几种？
2．什么是表达式？C++中有哪些常用的表达式？
3．表达式的值如何计算？表达式的类型如何确定？
4．什么是表达式语句，其与表达式有何区别？
5．什么是空语句？什么是复合语句？什么是块程序？
6．C++提供了哪几种循环语句？可以相互替代吗？可以相互嵌套吗？
7．C++中如何定义一个函数？如何声明一个函数？定义和声明有什么区别？
8．函数的类型是如何定义的？函数的值是什么？是否所有的函数都有返回呢？
9．如何定义一个整型数组？如何给一个数组赋值？
10．什么是指针？指针的值和类型如何定义？
11．各种常用的数据类型指针如何定义？
12．什么是类？如何声明一个类？其声明和实现部分分别是哪些？
13．类的成员访问控制关键字有几种？它们的区别是什么？
14．什么是构造函数？构造函数的特点是什么？
15．什么是析构函数？析构函数的特点是什么？
16．C++中的运算符可以重载吗？如何进行重载呢？

二、编程题

1．编程求下列公式的值：

$n^1+n^2+n^3+n^4+\cdots+n^{10}$，其中，n=1，2，3。

编写函数时，把函数的参数 n 设置为默认 2。

2．编程求出 Fibonacci 数列的第 n 项。Fibonacci 数列的特点是数列的前两项都是 1，而以后第一项都是前两项之和，例如：

```
1，1，2，3，5，8，…，n
```

3．从键盘上输入 5 个数，计算其平均值。要求用函数方式求解。

4．现在有 10 个人，求第 10 个人是多少岁？已知第 10 个人比第 9 个人大 2 岁，而第 9 个人的年龄也不知道，只知道他比第 8 个人大 2 岁。而第 8 个人的年龄也不知道，只知道比第 7 个人大 2 岁，依此类推。现只知道第 1 个人的年龄为 17 岁。

第 4 章　网络通信基础

在学习完前面一章后，相信读者已经掌握了 C++编程语言的各种特性以及基础语法。同时，对于使用 Visual C++来开发一些小程序已经没有问题。但是上面这些内容都不涉及网络。为此，在本章将主要帮助读者熟悉 Windows 网络编程的基础知识，并把这些知识同 Visual C++工具相结合，来实践开发一个简单的 Windows 网络应用程序。希望读者能够详细地阅读并实践。

本章主要涉及的内容如下：

- TCP/IP 协议的构成。
- Socket 的概念及其网络通信模式。
- Windows Sockets 类的介绍及使用。

4.1　TCP/IP 简介

本节主要简单介绍一下 TCP/IP 协议的内部结构，为后面的网络通信知识打下基础。TCP/IP 协议组之所以如此流行，大部分原因是其可以用在各种各样的信道和底层协议之中。例如 T1 和 X.25、以太网，以及 RS-232 串行接口之上。

确切地说，TCP/IP 协议是一组包括 TCP 协议和 IP 协议、UDP（User Datagram Protocol）协议、ICMP（Internet Control Message Protocol）协议及其他一些协议的协议组简称。

4.1.1　TCP/IP 整体构架概述

TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议）是 Internet 最基本的协议，简单地说，就是由底层的 IP 协议和 TCP 协议组成的。

在 Internet 没有形成之前，各个地方已经建立了很多小型的网络，称为局域网，Internet 的中文意义是“网际网”，其实际上就是将全球各地的局域网连接起来而形成的一个“网之间的网（即网际网）”。

然而，在连接之前的各式各样的局域网却存在不同的网络结构和数据传输规则，将这些小网连接起来后各网之间要通过什么样的规则来传输数据呢？这就像世界上有很多个国家，各个国家的人说各自的语言，世界上任意两个人要怎样才能互相沟通呢？如果全世界的人都能够说同一种语言（即世界语），这个问题不就解决了吗？TCP/IP 协议正是 Internet 上的“世界语”。TCP/IP 协议的开发工作始于 20 世纪 70 年代，是用于互联网的第一套协议。

说明： 实际上，TCP/IP 才是目前因特网范围内运行的唯一一种协议。

介绍 TCP/IP 协议的原理之前，先简单了解一下 OSI 参考模型。它是一种通信协议的 7 层抽象参考模型。在 OSI 的 7 层模型中，每一层执行某一特定任务。该模型的目的是使各种硬件在相同的层次上相互通信。

而 TCP/IP 协议并不完全符合 OSI 的 7 层参考模型。TCP/IP 通信协议采用了 4 层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求，如图 4.1 所示。

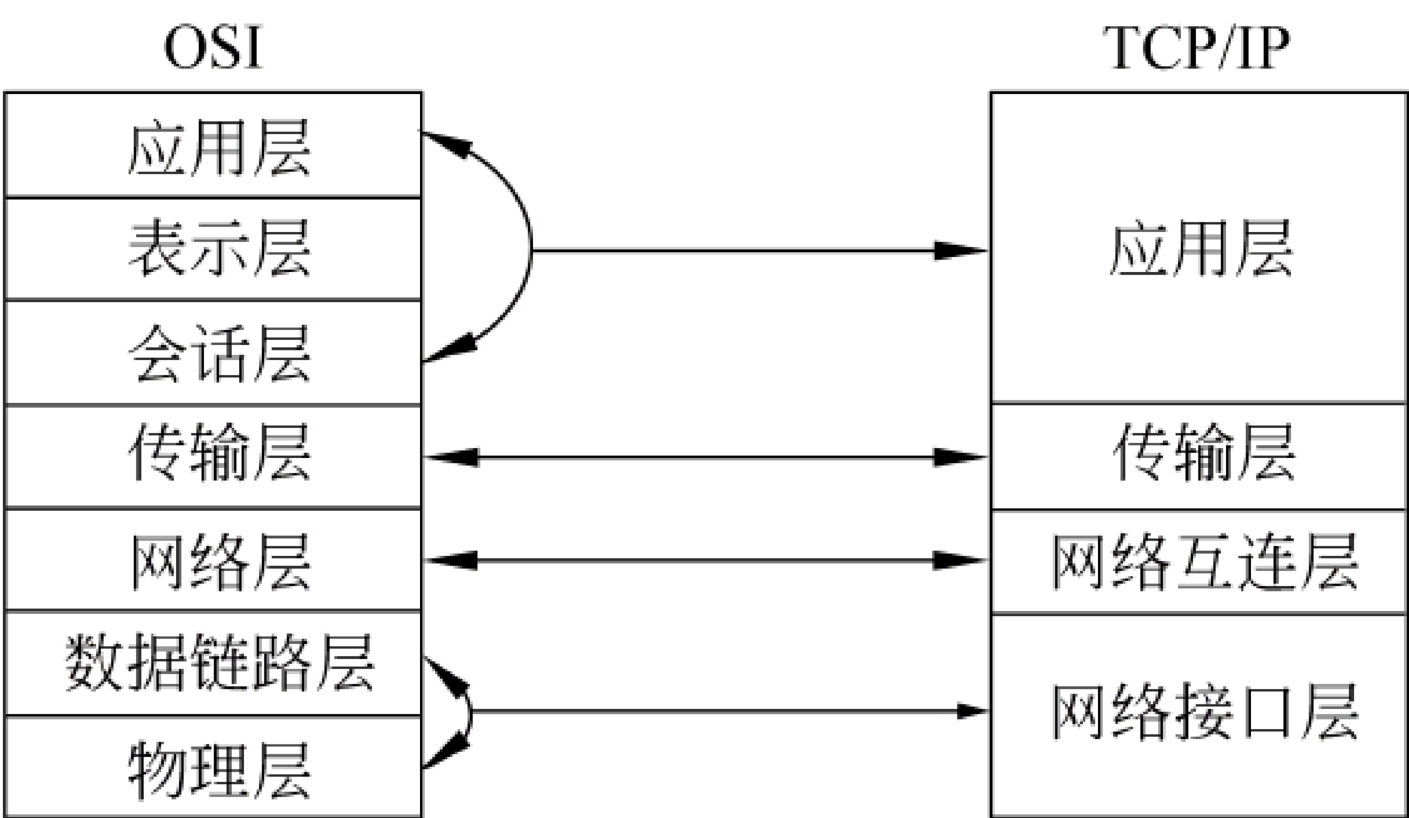


图 4.1　OSI 与 TCP/IP 对比

TCP/IP 协议的各层作用如下所述。

- 应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。
- 传输层：在此层中，它提供了结点间的数据传送服务，如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP 和 UDP 给数据包加入传输数据并把它传输到下一层中，这一层负责传送数据，并且确定数据已被送达并接收。
- 网络互连层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机（但不检查是否被正确接收），如网际协议（IP）。
- 网络接口层：实际上 TCP/IP 参考模型没有真正描述这一层的实现，只是要求能够提供给其上层——网络互连层一个访问接口，以便在其上传递 IP 分组。由于这一层次未被定义，所以其具体的实现方法将随着网络类型的不同而不同。

技巧：*在处理 Internet 网络中，只需要记住 TCP/IP 协议的分层就可以应用实际之中。*

4.1.2　TCP/IP 协议的应用

TCP/IP 协议可应用在任何互连网络上的通信，其可行性在许多地方都已经得到证实，包括家庭、校园、公司，以及全球 61 个国家实验室。例如在美国就有 National Science Foundation（NFS）、Department of Energy（DDE）、Department of Defense（DOD）、Health and Human Services Agency（HHS），以及 National Aeronautics and Space Administration（NASA）等大机构投注了相当大的资源来开发和应用 TCP/IP 网络。

这些技术的应用，让所有与网络相连的研究人员能够和全世界的同僚们共同分享资料和研究成果，感觉就像隔壁一样。Internet 互联网充分证明了 TCP/IP 的可行性及其优秀的整合性，使之能适应各种不同的现行网络技术。对今天的网络发展局面来说，TCP/IP 的应用可以说是一个卓越的成就。

TCP/IP 协议不仅成功的连接了不同网络，而且许多应用程序和概念也是完全以

TCP/IP 协议为基础发展出来，从而让不同的厂商能够忽略硬件结构开发出共同的应用程序，例如今天应用广泛的 WWW、E-mail、FTP、DNS 服务等。

4.1.3　TCP/IP 协议的特性

对于一个电子邮件的使用者来说，他无需透彻了解 TCP/IP 这个协议；但对于 TCP/IP 程式人员和网路管理人员来说，TCP/IP 的一些特性却是不能忽略的，如下所述。

1．Connectionless Packet Delivery Service（封包交换网络服务）

Conntectionless Packet Delivery Service 是整个网路服务的基础，几乎所有封包交换网络都提供这种服务。TCP/IP 根据信息中所含的位址资料来进行资料传送，不能确保每个独立路由的封包是可靠和依序地送达目的地。在每一个连线过程中，线路都不是被“独占”的，而是直接映射到硬体位址上，因此特别有效。更重要的是，此种封包交换方式的传送，使得 TCP/IP 能适应各种不同的网路硬体。

2．Reliable Stream Transport Service（可靠流传输服务）

因为 TCP/IP 中的封包交换并不能确保每一个封包的可靠性，因此就需要通信软件来自动侦测和修复传送过程中可能出现的错误，处理不良的封包。这种服务用来确保电脑程式之间能够建立连接和传送大量资料。关键的技术是将资料流进行切割，传送编号，然后通过接收方的确认（acknowledgement）来保证资料的完整性。

3．Network Technology Independent（独立网络技术）

在封包交换技术中，TCP/IP 是独立于硬体之上的。TCP/IP 有自己的一套资料包规则和定义，能应用在不同的网络之上。

4．Universal Interconnection（通用互连）

只要电脑用 TCP/IP 连接网络，都将获得一个独一无二的识别位址。资料包在交换的过程中，是以位址资料为依据的，不管封包所经过的路由选择如何，资料都能被送达指定的位址。

5．End-to-End Acknowledgements（端到端应答式）

TCP/IP 的确认模式是以“端到端”进行的。这样就无需理会封包交换过程中所参与的其他设备，发送端和接收端能相互确认才是我们关心的。

6．Application Protocol Standards（标准应用协议）

TCP/IP 除了提供基础的传送服务，还提供许多一般应用标准，让程序设计人员更有标准可依，而且也节省了许多不必要的重复开发。

正是由于 TCP/IP 协议具备了以上这些有利特性，才使得它在众多的网络连接协定中脱颖而出，成为大家喜爱和愿意遵守的标准。

4.2　TCP/IP 中的各种协议

在本节中，笔者就简单地介绍一下 TCP/IP 中的各种协议，以及其具备什么样的功能、是如何工作的。

4.2.1　IP 协议

IP（Internet Protocol）网际协议是 TCP/IP 协议的核心，也是网络层中最重要的协议。IP 协议是网络层协议，用在因特网上，TCP、UDP、ICMP、IGMP 数据都是按照 IP 数据格式发送的。

IP 协议是用于多个包交换连接起来的网络，其在源地址和目的地址之间传送数据报。IP 提供了对数据大小的重新组装功能，以适应不同网络对于包大小的要求。主要责任就是把数据从源地址传送到目的地址，并提供两个基本功能，即寻址和分段。IP 报文包的头结构如图 4.2 所示。

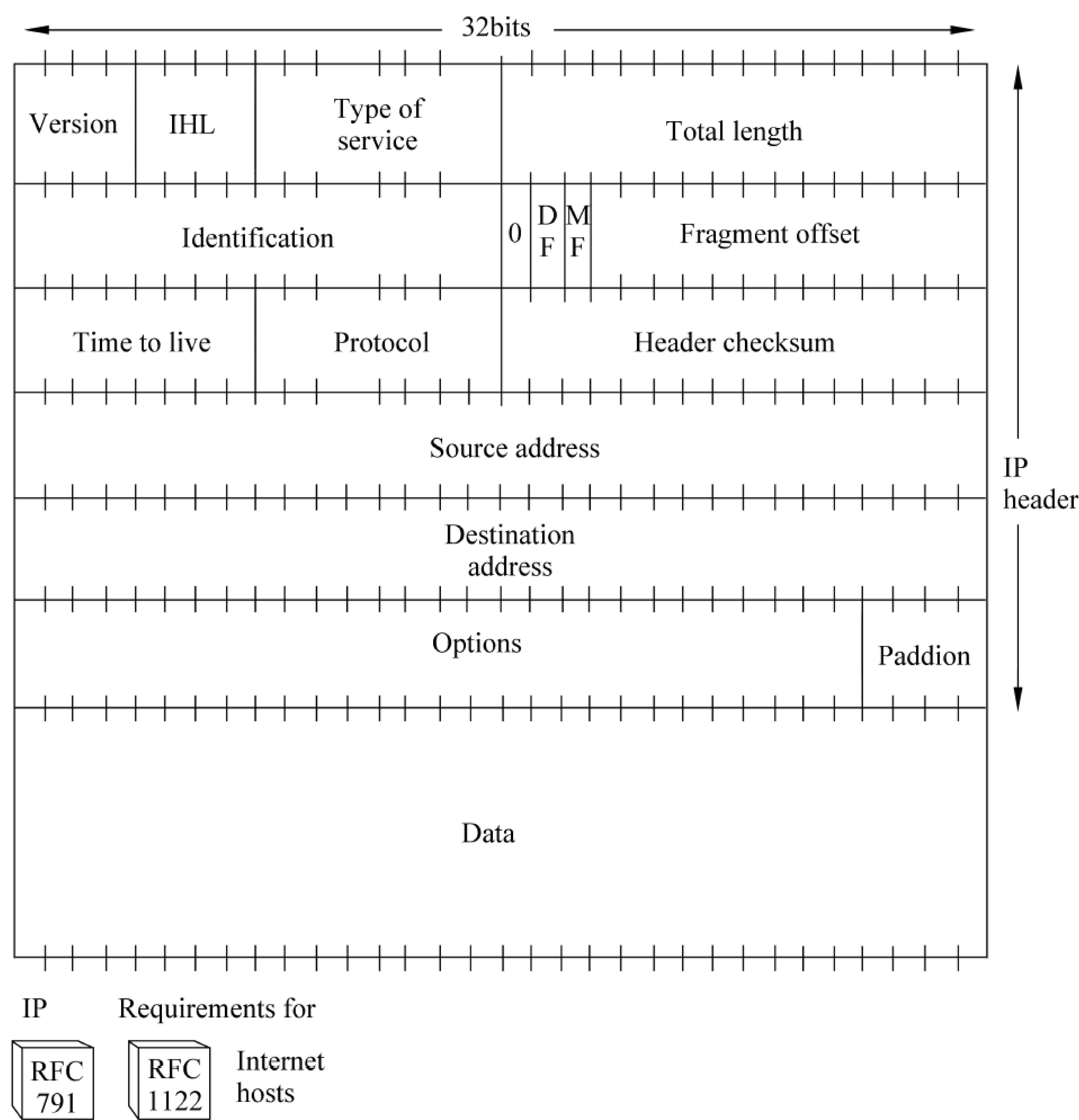


图 4.2　IP 报文结构

因为 IP 协议中并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏，其只

使用报头的校验码来校验数据的有效性。所以 IP 协议提供的是不可靠无连接的服务。此外 IP 还不负责流控制、包顺序和其他对于主机到主机协议来说很普通的服务。

IP 协议由主机到主机协议调用，这些协议负责调用本地网络协议将数据报传送到下一个网关或者目的主机。例如：TCP 可以调用 IP 协议，而在调用时传送目的地址和源地址作为参数，IP 形成数据报并调用本地网络（协议）接口传送数据报。

TCP/IP 中的数据传送一般由 IP 层接收从更低层（网络接口层，例如以太网设备驱动程序）发来的数据包，并把该数据包发送到更高层——TCP 或 UDP 层。相反，IP 层也把从 TCP 或 UDP 层接收来的数据包传送到更低层。IP 协议在整个协议体系中的位置如图 4.3 所示。

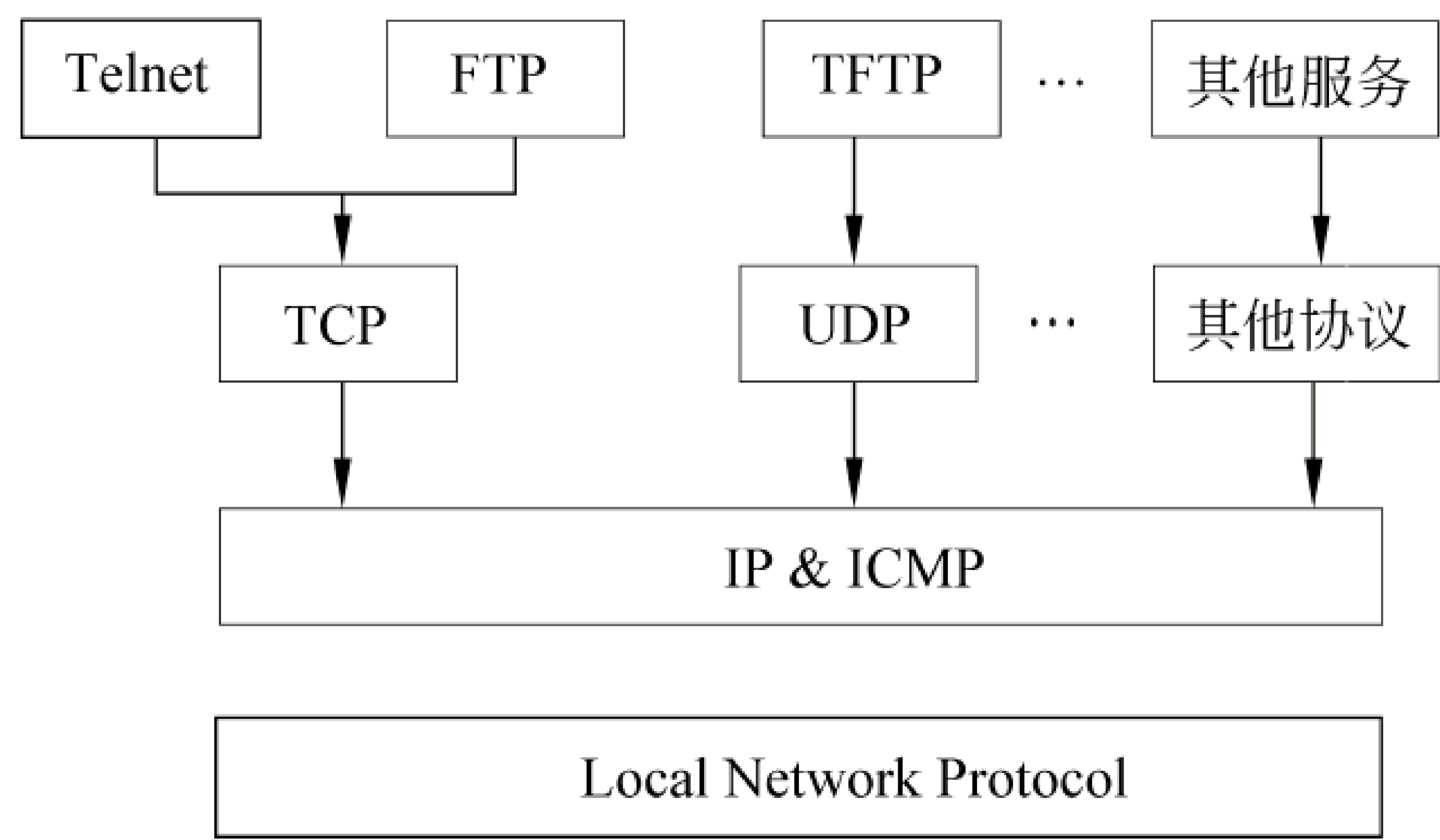


图 4.3　IP 协议与其他协议的关系

4.2.2　TCP 协议

TCP（Transmission Control Protocol）是面向连接的通信传输控制协议。其提供两台计算机之间的可靠无错的数据传输。应用程序利用 TCP 进行通信时，源和目标之间会建立一个虚拟连接。这个连接一旦建立，两台计算机之间就可以把数据当作一个双向的字节流进行交换。

TCP 协议主要是一种为了主机间实现高可靠的包交换传输协议。因为 TCP 将包排序并进行错误检查，同时实现虚电路间的连接。TCP 数据包中包括序号和确认，所以未按照顺序收到的包可以被排序，而损坏的包可以被重传。例如 Telnet、FTP、rlogin、X Windows 和 SMTP 这些应用都需要高度的可靠性。DNS 在某些情况下也使用 TCP（发送和接收域名数据库）。

整个 TCP 工作过程比较复杂，包括的内容如下所述。

（1）TCP 连接关闭：发送方主机和目的主机建立 TCP 连接并完成数据传输后，会发送一个将结束标记置 1 的数据包，以关闭这个 TCP 连接，并同时释放该连接占用的缓冲区空间。

（2）TCP 重置：TCP 允许在传输的过程中突然中断连接。

（3）TCP 数据排序和确认：在传输的过程中使用序列号和确认号来跟踪数据的接收情况。

（4）TCP 重传：在 TCP 的传输过程中，如果在重传超时时间内没有收到接收方主机对某数据包的确认回复，发送方主机就认为此数据包丢失，并再次发送这个数据包给接收方。

（5）TCP 延迟确认：TCP 并不总是在接收到数据后立即对其进行确认，其允许主机在接收数据的同时发送自己的确认信息给对方。

（6）TCP 数据保护（校验和）：TCP 是可靠传输的协议，其提供校验和计算来实现数据在传输过程中的完整性。

TCP 协议支持多种硬件构成的网络系统，其对于下层服务没有太多的要求，都假定下层只能提供不可靠的数据报服务。TCP 将收到的信息送到更高层的应用程序，例如 Telnet 的服务程序和客户程序。应用程序轮流将信息送回 TCP 层，TCP 层便又将这些数据向下传送到 IP 层、设备驱动程序和物理介质，最后到接收方。

TCP 的连接建立过程又称为 TCP 三次握手。首先发送方主机向接收方主机发起一个建立连接的同步（SYN）请求；接收方主机在收到这个请求后向发送方主机回复一个同步/确认（SYN/ACK）应答；发送方主机收到此包后再向接收方主机发送一个确认（ACK），此时 TCP 连接成功建立，如图 4.4 所示。

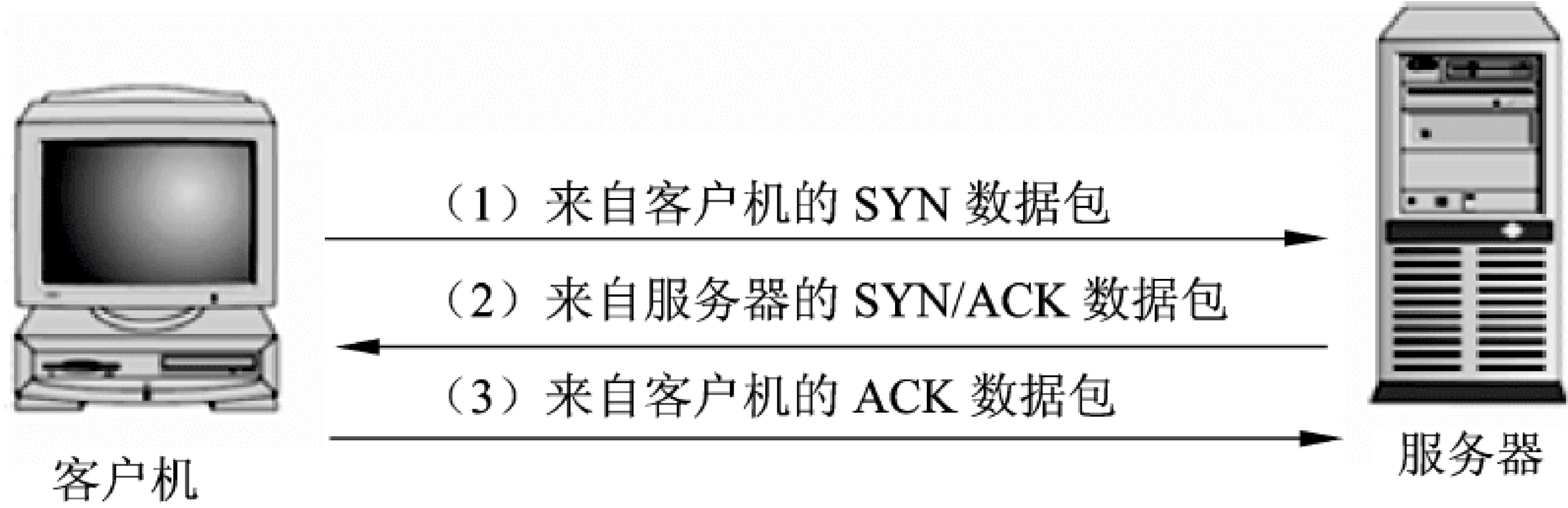


图 4.4　TCP 协议的三次握手过程

一旦初始的三次握手完成，在发送和接收主机之间将按顺序发送和确认段。关闭连接之前，TCP 使用类似的握手过程验证两个主机是否都完成发送和接收全部数据。

4.2.3　UDP 协议

UDP（User Datagram Protocol）是面向无连接通信的用户数据报协议。UDP 与 TCP 位于网络结构 4 层中的同一层，其对于数据包的顺序错误或重发是不检查的。因此，UDP 不保障可靠的数据传输，但能够向若干个目标发送数据，接收多个源地址的数据。

简单地说，如果一个客户机向服务器发送数据，这一数据会立即发出，不管服务器是否已准备接收数据。如果服务器收到客户机数据，它也不会确认收到与否。数据传送方法采用的是数据报格式。UPD 数据包的头部结构如图 4.5 所示。

源端口	目的端口
用户数据包的长度	校验和
数据包	

图 4.5　UDP 数据包的头部结构

又因为 UDP 的特性，所以其不被应用于那些使用虚电路的面向连接的服务，UDP 主要用于那些面向查询——应答的服务，例如 NFS。相对于 FTP 或 Telnet，这些服务需要交换的信息量较小。使用 UDP 的服务包括 NTP（网络时间协议）、DNS（DNS 也使用 TCP）

及各种视频通信协议。

🔔**技巧**：TCP 与 UDP 是应用最广的协议，主要区别是一个面向连接，一个不面向连接。

4.3　Socket 简介

TCP 和 UDP 服务通常有一个客户和服务器的关系，而这个网络关系是如何维持的呢？其实就是依靠 Socket 进行连接和维持的。

Socket 是一个软件结构，被客户程序或服务进程用来发送和接收信息。一个 Socket 对应一个 16 比特的数。服务进程通常使用一个固定的 Socket，例如，SMTP 使用 25、XWindow 使用 6000。下面我们就来学习什么是 Socket 及其如何使用的相关知识。

4.3.1　什么是 Sockets

所谓的 Socket，是指 TCP/IP 协议网络的 API。Socket 接口中定义了许多函数和例程，程序员可以用其来开发 TCP/IP 协议网络上的应用程序。要学 Internet 上 TCP/IP 协议的网络编程，必须理解 Socket 接口。

Socket 最先是在 Unix 操作系统里面的。如果了解 Unix 系统的输入和输出的话，就很容易了解 Socket 了。网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。Socket 也具有一个类似于打开文件的函数调用 Socket()，该函数返回一个整型的 Socket 套接字，随后的连接建立、数据传输等操作都是通过该 Socket 实现的。

4.3.2　Socket 网络通信模式

常用的 Socket 网络通信模式的类型有两种：流式 Socket（SOCK_STREAM）和数据报式 Socket（SOCK_DGRAM）。

- 流式 Socket（SOCK_STREAM）是一种面向连接的 Socket，其提供了双向、有序的、无重复的，以及无记录边界的数据流服务，适合处理大量数据。所以它是针对于面向连接的 TCP 服务应用的。
- 数据报式 Socket（SOCK_DGRAM）是一种无连接的 Socket，提供支持双向的数据流，它不保证传输数据的准确性，但保留了记录边界。它是针对于面向无连接的 UDP 服务应用的。

从程序员的角度来看，SOCK_STREAM 和 SOCK_DGRAM 这两类套接字涵盖了 TCP/IP 应用的全部，因为基于 TCP/IP 的应用，从协议栈的层次上讲，在传输层的确只可能建立于 TCP 或 UDP 协议之上，而 SOCK_STREAM 和 SOCK_DGRAM 又分别对应于 TCP 和 UDP，所以几乎所有的应用都可以用这两类套接字实现。

🔔**技巧**：在大多数程序中，Socket 就是网络通信的接口。不需要去明白网络的结构。

4.3.3　Socket 的函数

在程序中如果要使用 Socket，必须先建立一个 Socket。

1．建立 Socket

建立 Socket 可以调用 socket()函数，该函数返回一个类似于文件描述符的句柄。socket()函数原型为：

```
int socket(int domain, int type, int protocol);
```

其中，参数 domain 指明所使用的协议，通常为 PF_INET，表示是互联网协议，即 TCP/IP 协议；type 参数指定 Socket 的类型：SOCK_STREAM（TCP）或 SOCK_DGRAM（UDP），Socket 接口还定义了原始 Socket（SOCK_RAW），允许程序使用低层协议；protocol 通常赋值“0”。调用这个函数后，会返回一个整型 Socket 套接字，这个套接字将在后面用到。

Socket 套接字其实是一个指向内部数据结构的指针，其指向套接字表入口。调用 Socket()函数时，Socket 执行体将建立一个 Socket，实际上“建立一个 Socket”意味着为这个 Socket 数据结构分配存储空间。

两个网络程序之间的一个网络连接包括 5 种信息：通信协议、本地协议地址、本地主机端口、远端主机地址和远端协议端口。Socket 数据结构中包含这 5 种信息。

2．绑定 Socket

通过 Socket 调用返回一个 Socket 套接字后，在使用 Socket 进行网络传输以前，必须配置这个 Socket。面向连接的 Socket 客户端通过调用 connect()函数在 Socket 数据结构中保存本地和远端信息。无连接 Socket 的客户端和服务端，以及面向连接 Socket 的服务端通过调用 bind 函数来配置本地信息。

bind()函数将 Socket 与本机上的一个端口相关联，随后就可以在该端口上监听服务请求。bind()函数原型为：

```
int bind(int sockfd,struct sockaddr *MyAddr, int AddrLen);
```

其中，参数 Sockfd 是调用 Socket()函数返回的 Socket 套接字；MyAddr 是一个指向包含有本机 IP 地址及端口号等信息的 sockaddr 类型的指针；AddrLen 常被设置为 sizeof(struct sockaddr)。

struct sockaddr 结构类型是用来保存 Socket 信息的，其详情如下：

```
struct sockaddr {
unsigned short sa_family;   /*地址族， AF_xxx*/
char sa_data[14];           /*14 字节的协议地址*/
};
```

其中，填写 sa_family 一般为 AF_INET，代表这是 TCP/IP 地址；sa_data 则包含该 Socket 的 IP 地址和端口号。该地址结构随选择协议的不同而变化。所以是一个通用的 Socket 地址结构。

下面笔者再介绍一个与这个地址结构大小相同的 sockaddr_in 结构，这个结构更为常

用，因为其是用来专门标识 TCP/IP 协议下的地址结构的。

```
struct sockaddr_in {
short int sin_family;              /*地址族*/
unsigned short int sin_port;       /*端口号*/
struct in_addr sin_addr;           /*IP 地址*/
unsigned char sin_zero[8];         /*填充 0，以保持与 struct sockaddr 同样大小*/
};
```

其中，sin_family 必须设置为 AF_INET；sin_port 为服务端口，注意不要使用系统已固定的特殊端口，如 HTTP 服务的 80 端口；sin_addr 为一个 unsigned long 的 IP 地址；sin_zero 用来将 sockaddr_in 结构填充到与 struct sockaddr 同样的长度，可以用 memset()函数将其置为 0。

指向 sockaddr_in 的指针和指向 sockaddr 的指针可以相互转换，这意味着如果一个函数所需参数类型是 sockaddr 时，可以在函数调用的时候将一个指向 sockaddr_in 的指针转换为指向 sockaddr 的指针；或者相反。

使用 bind()函数时，可以用下面的赋值实现自动获得本机 IP 地址和随机获取一个没有被占用的端口号，例如：

```
MyAddr.sin_port = 0;                    /*系统随机选择一个未被使用的端口号*/
MyAddr.sin_addr.s_addr = INADDR_ANY;    /*填入本机 IP 地址*/
```

通过将 MyAddr.sin_port 置为 0，函数会自动选择一个未占用的端口来使用。同样，通过将 MyAddr.sin_addr.s_addr 置为 INADDR_ANY，系统会自动填入本机 IP 地址。

注意在使用 bind()函数时需要将 sin_port 和 sin_addr 转换成为网络字节优先顺序；而 sin_addr 则不需要转换。

技巧： 计算机数据存储有两种字节优先顺序：高位字节优先和低位字节优先。Internet 上数据以高位字节优先顺序在网络上传输，所以对于在内部是以低位字节优先方式存储数据的机器，在 Internet 上传输数据时就需要进行转换，否则就会出现数据不一致的情况。

在这里介绍几个字节顺序转换函数，如下所示。

```
htonl()：把 32 位值从主机字节序转换成网络字节序
htons()：把 16 位值从主机字节序转换成网络字节序
ntohl()：把 32 位值从网络字节序转换成主机字节序
ntohs()：把 16 位值从网络字节序转换成主机字节序
```

bind()函数在成功被调用时返回 0；出现错误时返回–1 并设置相应的错误号。需要注意的是，在调用 bind()函数时一般不要将端口号置为小于 1024 的值，因为 1～1024 是保留端口号，可以选择大于 1024 中的任何一个没有被占用的端口号。

3．客户端连接建立

面向连接的客户端程序可以使用 Connect 函数来配置 Socket，并与远端服务器建立一个 TCP 连接，其函数原型为：

```
int connect(int sockfd, struct sockaddr *serv_addr,int addrlen);
```

其中，sockfd 是 socket()函数返回的 socket 套接字；serv_addr 是包含远端主机 IP 地址和端口号的指针；addrlen 是远端地址结构的长度。Connect 函数在调用错误时返回–1 并设置相应的错误值。

注意：在进行客户端程序时，无需调用 bind()函数，因为这种情况下只需知道目的机器的 IP 地址，而客户端程序通过哪个端口与服务器建立连接调用者并不需要关心，connect 函数会为程序自动选择一个未被占用的端口，并通知调用者数据什么时候会到达这个端口。

connect()函数启动和远端主机的直接连接。只有面向连接的客户程序使用 Socket 时才需要将此 Socket 与远端主机相连。无连接协议从不建立直接连接。面向连接的服务器也从不启动一个连接，它只是被动地在协议端口监听客户的请求。

4．服务器端的监听

服务器端程序调用 listen()函数使 Socket 处于被动的监听模式，并为该 Socket 建立一个输入数据队列，将到达的服务请求保存在此队列中，直到程序处理。listen()函数的原型如下：

```
int listen(int sockfd， int backlog);
```

其中，sockfd 是 Socket 系统调用返回的 Socket 套接字；backlog 指定在请求队列中允许的最大请求数，进入的连接请求将在队列中等待 accept（将在后面进行讲解）。

Backlog 对队列中等待服务请求的数目进行了限制，大多数系统默认值为 20。如果一个服务请求到来时，输入队列已满，该 Socket 将拒绝服务连接请求，客户端收到一个出错信息。当调用该函数出现错误时函数返回–1 并设置相应的错误值。

5．服务器端的接收

在建立好输入队列后，服务器就调用 accept()函数，然后睡眠并等待客户的连接请求。当有客户连接时，服务器端程序通过调用 accept()函数让服务器接收客户的连接请求。accept()函数的原型如下：

```
int accept(int sockfd, void *addr, int *addrlen);
```

其中，sockfd 是被监听的 Socket 套接字；addr 通常是一个指向 sockaddr_in 变量的指针，该变量用来存放提出连接请求服务的主机信息（某台主机从某个端口发出该请求）；addrlen 通常为一个指向值为 sizeof(struct sockaddr_in)的整型指针变量。当出现错误时 accept()函数返回–1 并设置相应的错误值。

当 accept 接收函数监视的 Socket 收到一个服务连接请求时，Socket 将建立一个新的 Socket，并把这个新的 Socket 和请求连接进程的地址联系起来，收到服务请求的初始 Socket 仍可以继续在以前的 Socket 上监听，同时可以在新的 Socket 套接字上进行数据传输操作。

6．面向连接的数据发送

在 Socket 中，可以利用 send()函数用于面向连接的 Socket 上进行数据发送。send()函

数的原型如下：

```
int send(int sockfd, const void *msg, int len, int flags);
```

参数 sockfd 是用来传输数据的 Socket 套接字；msg 是一个指向要发送数据的指针；len 是以字节为单位的数据的长度；flags 一般情况下置为 0。

send()函数调用后返回实际上发送出的字节数，可能会少于希望发送的数据。在程序中应该将 send()的返回值与将要发送的字节数进行比较。当 send()返回值与 len 不匹配时，应该对这种情况进行处理，如代码 4.1 所示。

代码 4.1　完整发送全部数据

```
char *msg = "Hello World!";                    //初始化字符串
int nlen = strlen(msg);                        //得到字符串长度
int nslen = 0;                                 //用于保存已经发送的长度
while(nslen < nlen)                            //判断是否发送完毕
{                                              //循环发送，直到发送完毕为止
    nslen += send(sock, msg+nslen, (nlen-nslen), 0);
}
```

7．面向连接的数据接收

在 Socket 中，可以利用 recv()函数用于面向连接的 Socket 上进行数据接收，其函数原型如下：

```
int recv(int sockfd,void *buf,int len,unsigned int flags);
```

其中，sockfd 是接收数据的 Socket 套接字；buf 是存放接收数据的缓冲区；len 是缓冲的长度。flags 也被置为 0。Recv()返回实际上接收的字节数，当出现错误时，返回–1 并设置相应的错误值。

8．无连接的数据发送

sendto()函数用于在无连接的数据报 Socket 方式下进行数据发送。由于本地 Socket 并没有与远端的主机建立连接，所以在发送数据时应指明目的地址。sendto()的函数原型如下：

```
int sendto(int sockfd, const void *msg,int len,unsigned int flags,const
struct sockaddr *to, int tolen);
```

这个函数只比 send()函数多了两个参数，其中参数 to 表示目的机的 IP 地址和端口号信息，而 tolen 常常被赋值为 sizeof(struct sockaddr)。sendto()函数也返回实际发送的数据字节长度或在出现发送错误时返回–1。

9．无连接的数据接收

recvfrom()函数用于在无连接的数据报 Socket 方式下进行数据接收。recvfrom()函数原型如下：

```
int recvfrom(int sockfd,void *buf,int len,unsigned int flags,struct sockaddr
*from,int *fromlen);
```

其中，from 是一个 struct sockaddr 类型的变量，该变量保存源机的 IP 地址及端口号。fromlen 一般设置为 sizeof (struct sockaddr)。当 recvfrom()返回时，fromlen 包含实际存入 from 中的数据字节数。recvfrom()函数返回接收到的字节数或当出现错误时返回–1，并设置相应的错误值。

如果对数据报类型的 Socket 调用了 connect()函数时，其实也可以利用 send()和 recv()进行数据传输，但该 Socket 仍然是数据报类型 Socket，并且只会利用传输层的 UDP 服务。但在发送或接收数据报时，内核会自动为之加上目的地和源地址信息。

10．域名转换为 IP 地址

由于 IP 地址难以记忆和读写，所以为了方便，人们常常用域名来表示主机，这就需要进行域名和 IP 地址的转换。函数 gethostbyname()可以完成域名转换，其函数的原型如下：

```
struct hostent *gethostbyname(const char *name);
```

这个函数返回值为 hostent 的结构类型，其定义如下：

```
struct hostent {
char *h_name;                 /*主机的官方域名*/
char **h_aliases;             /*一个以 NULL 结尾的主机别名数组*/
int h_addrtype;               /*返回的地址类型，在 Internet 环境下为 AF-INET*/
int h_length;                 /*地址的字节长度*/
char **h_addr_list;           /*一个以 0 结尾的数组，包含该主机的所有地址*/
};
```

当 gethostname()函数调用成功时，返回指向 struct hostent 的指针，当调用失败时返回–1。

11．关闭端口

当所有的数据操作结束以后，可以调用 close()函数来释放该 Socket，从而停止在该 Socket 上的任何数据操作。该函数的原型如下：

```
close(sockfd);
```

其中，sockfd 是打开的 Socket 套接字。其实也可以调用 shutdown()函数来关闭该 Socket。该函数允许只停止在某个方向上的数据传输，而一个方向上的数据传输继续进行。

例如：可以关闭某 Socket 的写操作而允许继续在该 Socket 上接收数据，直至读入所有数据。其函数原型如下：

```
int shutdown(int sockfd,int how);
```

其中，sockfd 是需要关闭的 Socket 的套接字。参数 how 允许为 shutdown 操作选择。有如下几种方式。

- 0：不允许继续接收数据；
- 1：不允许继续发送数据；
- 2：不允许继续发送和接收数据。

shutdown 在操作成功时返回 0，在出现错误时返回–1 并设置相应错误值。

技巧：打开后的 Socket 要记得关闭，否则会造成资源浪费。

4.3.4　Socket 的使用示例

在这一节中，笔者将把前面的函数进行综合，并给出详细示例。现在要求服务器通过 Socket 连接，向客户端发送字符串 “You are Welcome!”。只要在服务器上运行该服务器软件，在客户端运行客户软件，客户端就会收到该字符串。其示例代码如代码 4.2 所示。

代码 4.2　面向连接的服务器端 Socket 示例

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <errno.h>
04  #include <string.h>
05  #include <winsock.h>
06  #define SERVPORT 3333    /*服务器监听端口号*/
07  #define BACKLOG 10       /*最大同时连接请求数*/
08
09  void main()
10  {
11      int sockfd,client_fd;
                    /*sock_fd: 监听 socket; client_fd: 数据传输 socket*/
12      struct sockaddr_in my_addr;          /*本机地址信息*/
13      struct sockaddr_in remote_addr;      /*客户端地址信息*/
14      char * msg = "You are Welcome!\n";
15      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                                             /*建立面向连接的 socket*/
16          printf("socket 创建出错！ ");
17          exit(1);
18      }
19      my_addr.sin_family=AF_INET;      //各种参数的初始化
20      my_addr.sin_port=htons(SERVPORT);
21      my_addr.sin_addr.s_addr = INADDR_ANY;
22      memset(&(my_addr.sin_zero),0, 8);
23                                       //绑定端口
24      if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
        sockaddr)) == -1) {
25          printf("bind 出错");
26          exit(1);
27      }
28
29      if (listen(sockfd, BACKLOG) == -1) {//监听端口
30          printf("listen 出错");
31          exit(1);
32      }
33      while(1) {
34          sin_size = sizeof(struct sockaddr_in);
35          //接收用户连接
36         if ((client_fd = accept(sockfd, (struct sockaddr *)&remote_addr,
            &sin_size)) == -1) {
37              printf("accept 出错");
38              continue;
39          }
40          //发送数据到客户端
41          if (send(client_fd, msg, strlen(msg), 0) == -1){
42              printf("send 出错");
```

```
43              close(client_fd);
44              exit(0);
45          }
46          close(client_fd);        //关闭端口
47      }
48  }
```

代码解析：代码中几个重点加粗的语句就是实现服务器端的工作流程，其步骤如下所述。

（1）第 15 行调用 socket()函数创建一个 Socket。

（2）第 24 行调用 bind()函数将其与本机地址及一个本地端口号绑定。

（3）第 29 行调用 listen()函数在相应的 socket 上监听。

（4）第 36 行当 accpet()函数接收到一个连接服务请求时，将生成一个新的 Socket。

（5）第 41 行调用 send()函数向客户端发送字符串“You are welcome!”。

（6）第 46 行最后关闭该 Socket。

学习了服务器端的代码后，再来看看客户端对 Socket 的应用，其如代码 4.3 所示，

代码 4.3　面向连接的客户端 Socket 示例

```
01  #include<stdio.h>
02  #include <stdlib.h>
03  #include <errno.h>
04  #include <string.h>
05  #include <winsock.h>
06  #define SERVPORT 3333
07  #define MAXDATASIZE 100                    /*每次最大数据传输量*/
08  void main(int argc, char *argv[])
09  {
10      int sockfd, recvbytes;
11      char buf[MAXDATASIZE] = {0};           //接收数据缓冲区
12      struct hostent *host;
13      struct sockaddr_in serv_addr;          //主机地址
14      if (argc < 2) {                        //当无参数时，提示出错
15          printf(stderr,"Please enter the server's hostname!\n");
16          exit(1);
17      }
18                                             //通过域名转换为 IP 地址
19      if((host=gethostbyname(argv[1]))==NULL) {
20          printf("gethostbyname 出错");
21          exit(1);
22      }
23                                             //创建 socket
24      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
25          printf("socket 创建出错");
26          exit(1);
27      }
28      serv_addr.sin_family=AF_INET;          //初始化参数
29      serv_addr.sin_port=htons(SERVPORT);
30      serv_addr.sin_addr = *((struct in_addr *)host->h_addr);
31      memset(&(serv_addr.sin_zero),0,8);
32                                             //连接服务器
33      if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(struct
        sockaddr)) == -1) {
34      printf("connect 出错");
35      exit(1);
36      }
```

```
37                                              //接收服务器数据
38      if ((recvbytes=recv(sockfd, buf, MAXDATASIZE, 0)) ==-1) {
39      printf("recv 出错");
40      exit(1);
41      }
42      buf[recvbytes] = '\0';
43      printf("Received: %s",buf);
44      close(sockfd);                          //关闭服务器
45  }
```

代码解析：代码中几个重点加粗的语句就是实现客户端程序，其执行流程如下所述。

（1）第 19 行通过服务器域名获得服务器的 IP 地址并创建一个 Socket。

（2）第 24 行调用 socket()函数创建一个 Socket。

（3）第 33 行调用 connect()函数与服务器建立连接。

（4）第 38 行连接成功之后调用 recv()函数接收从服务器发送过来的数据。

（5）第 44 行最后关闭 Socket。

无连接的客户/服务器程序在原理上和连接的客户/服务器是一样的，这里就不再举例。两者的区别在于无连接的客户/服务器中的客户一般不需要建立连接，而且在发送接收数据时，需要指定远端机的地址。

4.4　Windows CSockets 类的介绍及使用

前面介绍的是 WinSock API 函数的网络应用开发，它不是基于类为主的开发。为此，Microsoft 公司在 MFC 中，把前面介绍的复杂的 WinSock API 函数封装到类里，这使得编写网络应用程序更容易。这两个类分别是：CAsyncSocket 类和 CSocket 类。

4.4.1　CAsyncSocket 类和 CSocket 类的介绍

CAsyncSocket 类封装了 WinSock API 函数，提供基于异步通信的套接字服务，为高级网络程序员带来更加有力而灵活的方法。这个类基于程序员了解网络通信的假设，目的是为了在 MFC 中使用 WinSock，程序员有责任处理诸如阻塞、字节顺序和转换字符的任务。

为了给程序员提供更方便的接口以自动处理这些任务，MFC 又派生出了 CSocket 类，这个类是由 CAsyncSocket 类继承下来的，其提供了比 CAsyncSocket 更高层的 WinSock API 接口。

例如，CSocket 类可以将套接字上发送和接收的数据和一个文件对象（CSocketFile 类）关联起来，通过读写文件来达到发送和接收数据的目的。CSocket 类还可以与 CArchive 类一起合作来管理发送和接收的数据，这使管理数据收发更加便利。

CSocket 提供的通信模式为阻塞模式，数据未接收到或者未发送完之前调用都不会返回。这对于 CArchive 的同步操作是至关重要的。此外，通过 MFC 类来设计网络通信应用时，可以不考虑网络 Byte 顺序和很多通信细节。

在一次网络通信/连接时，只需要设置以下几个参数就可以使用。

- 本地 IP 地址。

- 本地端口。
- 对方端口号。
- 对应 IP 地址。

当设置完这几个参数后，就建立起一个全关联的套接字。在这个套接字上可以双向交换数据。如果使用无连接（UDP）通信时，只需要设置前两个参数在这个套接字上，即建立一个半关联。在发送和接收时指明另一半的参数就可以了。

说明：不论是面向连接（TCP）还是无连接（UDP）的通信，双方的工作端口不需要相同，如服务器可以是 8000，而客户端可以是 2000。

事实上，在 MFC 中 CAsyncSocket 逐个封装了 WinSock API，每个 CAsyncSocket 对象代表一个 Windows Socket 对象，使用 CAsyncSocket 类要求程序员对网络编程较为熟悉。

而 CSocket 对象提供阻塞模式，因为阻塞功能对于 CArchive 的同步操作是至关重要的。

4.4.2　阻塞和非阻塞模式

一个 Socket 可以处于“阻塞模式”或“非阻塞模式”。当一个套接字处于阻塞模式（即同步操作）时，其阻塞函数直到操作完成才会返回控制权，之所以称为阻塞是因为此套接字的阻塞函数在完成操作返回之前什么也做不了。

如果一个 Socket 处于非阻塞模式（即异步操作）时，则会立即返回。在 CAsyncSocket 类中可以用 GetLastError 成员函数查询最后的错误，如果错误是 WSAEWOULDBLOCK 则说明有阻塞，而 CSocket 绝不会返回 WSAEWOULDBLOCK，因为其自己管理阻塞。

微软建议尽量使用非阻塞模式，通过网络事件的发生而通知应用程序进行相应的处理。但在 CSocket 类中，为了利用 CArchive 处理通信中的许多问题和简化编程，它的一些成员函数总是具有阻塞性质的，这是因为 CArchive 类需要同步的操作。

在 Win32 环境下，如果要使用具有阻塞性质的套接字，应该放在独立的工作线程中处理，利用多线程的方法使阻塞不至于干扰其他线程，也不会把 CPU 时间浪费在阻塞上。多线程的方法既可以使程序员享受 CSocket 带来的简化编程的便利，又不会影响用户界面对用户的反应。

4.4.3　类的成员函数介绍

在这里，笔者主要介绍一下 CAsyncSocket 类的成员函数。因为 CSocket 类是继承于这个类的，所以二者的用法大致上也是相同的，只是实现不同而已。

1．创建 CAsyncSocket 对象

要创建 CAsyncSocket 对象时，可以调用其 Create 成员函数。该成员函数原型如下：

```
BOOL CAsyncSocket::Create(
    UINT nSocketPort =0,            /*自动分配一个端口号*/
    int nSocketType = SOCK_STREAM,  /*TCP 连接*/
    long lEvent = FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT | FD_CONNECT |
    FD_CLOSE,
```

```
    LPCTSTR lpszSocketAddress = NULL
)
```

其中，nSocketPort 是指定使用的端口号，如果是服务方，则使用一个众所周知的端口供服务方连接；如果是客户方，典型做法是接受默认参数，使套接字可以自主选择一个可用端口通常在客户端使用（即设置为 0）；nSocketType 是指定使用的通信协议；lpszSocketAddress 是指定本地 IP 地址，可以使用分点法或者域名来表示，例如：192.168.1.10 和 www.baidu.com。

可以通过指明 lEvent 所包含的标记来确定需要异步处理的事件。对于指明的相关事件的相关函数调用都不需要等待完成后才返回，函数会马上返回。然后在完成任务后发送事情通知，并利用重载以下成员函数来处理各种网络事件，如表 4.1 所示。

表 4.1　创建端口时指定的事件和需要重载函数

参　数　值	事　　件	重载的函数
FD_READ	有数据到达	void OnReceive(int nErrorCode)
FD_WRITE	有数据发送	void OnSend(int nErrorCode)
FD_OOB	收到外带数据	void OnOutOfBandData(int nErrorCode)
FD_ACCEPT	服务器端等待连接成功	void OnAccept(int nErrorCode)
FD_CONNECT	客户端连接成功	void OnConnect(int nErrorCode)
FD_CLOSE	套接字关闭	void OnClose(int nErrorCode)

在重载函数被调用时，其参数 nErrorCode 会被设置一个值。当这个值为 0 时，表示正常完成；如果为非零时，表示有错误。可以通过 CAsyncSocket 的成员函数 GetLastError()得到相应的错误值。

2．绑定端口

在创建完 CAsyncSocket 对象后，如果是服务器端，就需要绑定一个端口来等待客户端连接，绑定端口的成员函数如下：

```
BOOL CAsyncSocket::Bind(
    UINT nSocketPort =0,                    /*自动分配一个端口号*/
    LPCTSTR lpszSocketAddress = NULL
)
```

其中，nSocketPort 为想要绑定的端口号，绑定成功后就可以在该端口上监听服务请求。lpszSocketAddress 是指定本地 IP 地址，方法同 Create()函数。

3．监听端口

当服务器端口绑定成功后，就可以在该端口上监听来自客户端的服务请求。监听端口的成员函数原型如下：

```
BOOL CAsyncSocket::Listen(
    int nConnectionBacklog=5                    /*最大同时连接数*/
)
```

其中，nConnectionBacklog 用于指定同时可以接受的最大连接数，但要注意，这里不是说总共可以接受的客户端连接数量，而是同时连接的最大数量。

4. 接受连接

在监听端口收到有服务连接请求时，服务器就可以调用 CAsyncSocket 类的 Accept 成员函数来响应这个连接请求，该成员函数的原型如下：

```
BOOL CAsyncSocket::Accept(
    CAsyncSocket& rConnectedSocket,
    SOCKADDR * lpSockAddr = NULL,
    int * lpSockAddrLen = NULL
)
```

其中，rConnectedSocket 是套接字变量的引用，调用该函数时，函数会自动创建一个新的套接字，并把这个套接字的引用返回给调用者。调用者可以通过这个端口与这个连接进行通信。lpSockAddr 用来存放提出连接请求服务的主机的信息，如果为空，则表示调用者不需要知道。lpSockAddrLen 表示 SOCKADDR 地址结构长度。

5. 连接服务器端口

作为客户端，需要发起连接与服务器进行通信。发起连接可以调用 CAsyncSocket 类的成员函数 Connect()，其函数原型如下：

```
BOOL CAsyncSocket::Connect(
     LPCTSTR lpszSocketAddress,          /*服务器 IP 地址*/
     UINT nHostPort                      /*端口号*/
)
```

其中，lpszSocketAddress 指定想要连接服务器的 IP 地址，可以用分点法或者域名法（其表示同 Create）。nHostPort 指定需要连接到的服务器端口号。

6. 发送数据

当服务器与客户端连接成功后就可以发送数据。发送数据可以调用 CAsyncSocket 类的成员函数 Send()。其函数原型如下：

```
int CAsyncSocket::Send (
    const void * lpBuf,                  /*指向需要发送数据的存储区首地址*/
    int nBufLen,                         /*需发送数据的长度*/
    int nFlags=0                         /*一般设置为 0*/
)
```

其中，lpBuf 为指向需要发送数据的存储区域的指针。nBufLen 是需要发送的数据长度。flags 一般情况下置为 0。

如果是无连接的 UDP 报文，发送数据可以调用 CAsyncSocket 类的成员函数 SendTo()，其函数原型如下：

```
int CAsyncSocket::SendTo (
    const void * lpBuf,                  /*指向需要发送数据的存储区首地址*/
    int nBufLen,                         /*需发送数据的长度*/
    UINT nHostPort,                      /*目的主机端口*/
    LPCTSTR lpszHostAddress = NULL,      /*目的主机地址*/
```

```
    int nFlags=0                          /*一般设置为 0*/
)
```

其中，nHostPort 指定数据发送到的目的主机端口。lpszHostAddress 指定数据发送到的目的地址。

这两个函数的返回值为已经发送的数据长度。当为–1 时，可以通过 CAsyncSocket 的成员函数 GetLastError()得到相应的错误值。

7. 接收数据

既然有数据的发送，自然就有接收，要接收数据可以调用 CAsyncSocket 类的成员函数 Receive()，其函数原型如下：

```
int CAsyncSocket::Receive (
    const void * lpBuf,                   /*指向接收数据的存储缓冲区指针*/
    int nBufLen,                          /*存储缓冲区的最大长度*/
    int nFlags=0                          /*一般设置为 0*/
)
```

其中，lpBuf 为指向接收数据的存储区的指针。nBufLen 是存储缓冲区的最大长度。flags 一般情况下置为 0。

如果是接收无连接的 UDP 报文，接收数据可以调用 CAsyncSocket 类的成员函数 ReceiveFrom ()，其函数原型如下：

```
int CAsyncSocket:: ReceiveFrom (
    const void * lpBuf,                   /*指向接收数据的存储缓冲区地址*/
    int nBufLen,                          /*接收缓冲区最大长度 */
    CString &rSocketAddress,              /*接收数据主机的地址信息*/
    UINT& rSocketPort,                    /*接收数据主机的端口信息*/
    int nFlags=0                          /*一般设置为 0*/
)
```

其中，rSocketAddress 存储发送数据的源主机地址。rSocketPort 存储发送数据的源主机端口。

这两个函数的返回值为已经接收到的数据长度。当为–1 时，可以通过 CAsyncSocket 的成员函数 GetLastError()得到相应的错误值。

8. 关闭端口

当通信结束时，需要调用 Close 成员函数来关闭连接端口，其成员函数的原型如下：

```
BOOL CAsyncSocket::Close()
```

在调用以上这些成员函数时，都有一个布尔类型的返回值来表明是否调用成功。当发生错误时，可以通过 CAsyncSocket 的 GetLastError()函数来查看错误值。

现在，笔者再把整个面向连接的服务器与客户端通信过程介绍一下，其如图 4.6 所示。

由于 CSocket 是从 CAsyncSocket 类派生的，所以其拥有 CAsyncSocket 类的所有公有成员函数和功能，这里就不再详细介绍。只是要注意，要创建 CSocket 的对象时，只能调用 CSocket 类的创建函数。其函数原型如下：

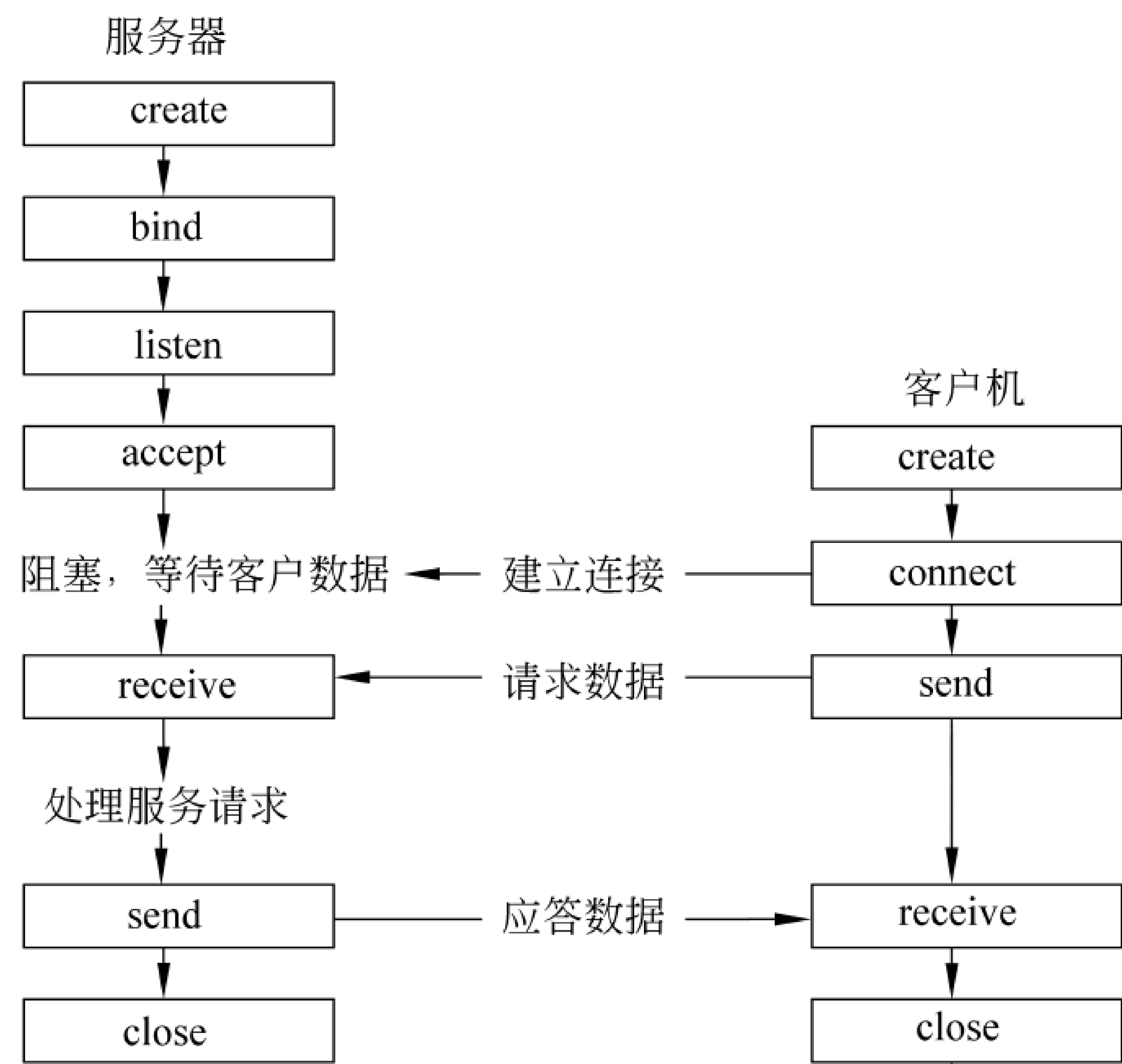


图 4.6　面向连接的通信流程

```
BOOL CAsyncSocket::Create(
    UINT nSocketPort =0,                /*自动分配一个端口号*/
    int nSocketType = SOCK_STREAM,      /*TCP 连接*/
    LPCTSTR lpszSocketAddress = NULL
)
```

这样创建出来的套接字，是不支持异步处理事件（非阻塞模式）的，所有成员函数的调用都必须完成后才会返回给调用者。

4.4.4　CAsyncSocket 和 CSocket 类的编程模型

相对于 CAsyncSocket 和 CSocket 类的不同的工作方式，其编程模型也不同。笔者在这里分别对其进行简单介绍。

1．CAsyncSocket 类的编程模型

在一个 MFC 应用程序中，要想轻松处理多个网络协议，而又不失灵活性时，可以考虑使用 CAsyncSocket 类，其效率比 CSocket 类要高。

CAsyncSocket 类针对字节流型套接字的编程模型流程如下所述：

（1）构造一个 CAsyncSocket 对象，并用这个对象的 Create 成员函数产生一个 Socket 句柄。可以按如下两种方法构造：

```
CAsyncSocket sock;                       //使用默认参数产生一个字节流套接字
Sock.Create();
```

或在指定端口号产生一个数据报套接字

```
CAsyncSocket*pSocket=newCAsyncSocket;
int nPort=80;
pSocket->Create(nPort,SOCK-DGRAM);             //建立端口
```

上面的第一种方法是在栈上产生一个 CAsyncSocket 对象，而第二种方法是在堆上产生 CAsyncSocket 对象；第一种方法中 Create 成员函数用默认参数产生一个字节流套接字，第二种方法中用 Create 成员函数在指定的端口产生一个数字报套接字。

（2）如是客户方程序，用 CAsyncSocket∷Connect 成员函数连接到服务方；如是服务方程序，用 CAsyncSocket∷Listen 成员函数开始监听，一旦收到连接请求，则调用 CAsyncSocket∷Accept 成员函数开始接收。

注意：CAsyncSocket∷Accept 成员函数要用一个新的并且是空的 CAsyncSocket 对象作为其参数，这里所说的“空的”，是指这个新对象还没有调用 Create 成员函数。

（3）调用其他的 CAsyncSocket 类的 Receive、ReceiveFrom、Send 和 SendTo 等成员函数进行数据通信。

（4）通信结束后，销毁 CAsyncSocket 对象。如果是在栈上产生的 CAsyncSocket 对象，则对象超出定义的范围时自动被析构；如果是在堆上产生，也就是用了 new 这个操作符，则必须使用 delete 操作符销毁 CAsyncSocket 对象。

2. CSocket 类编程模型

使用 CSocket 对象涉及 CArchive 和 CSocketFile 类对象。以下介绍的针对字节流型套接字的操作步骤中，只有第 3 步对于客户方和服务方操作是不同的，其他步骤都相同。

（1）构造一个 CSocket 对象。

（2）使用这个对象的 Create 成员函数产生 socket 对象。在客户方程序中，除非需要数据报套接字，Create()函数一般情况下应该使用默认参数。而对于服务方程序，必须在调用 Create 时指定一个端口。

（3）如果是客户方套接字，则调用 CAsyncSocket 的成员函数 Connect()与服务方套接字连接；如果是服务方套接字，则调用 Listen 开始监听来自客户方的连接请求，收到连接请求后，调用 Accept()函数接受请求，建立连接。请注意 Accept 成员函数需要一个新的并且为空的 CSocket 对象作为其参数（前面已经解释过）。

（4）产生一个 CSocketFile 对象，并把其与 CSocket 对象关联起来。

（5）为接收和发送数据各产生一个 CArchive 对象，把其与 CSocketFile 对象关联起来。切记 CArchive 是不能和数据报套接字一起工作的。

（6）使用 CArchive 对象的 Read()、Write()等函数在客户与服务方传送数据。

（7）通信完毕后，销毁 CArchive、CSocketFile 和 CSocket 对象

4.5　CAsyncSocket 类综合应用

学习了前面的知识，现在把这些知识结合起来，开发一个使用 CAsyncSocket 类的简单网络应用程序。整个应用程序要求如下：

（1）设定服务器的服务端口为 10000，采用 TCP 连接方式进行。当有客户端连接时，在收到的信息编辑框中提示客户端 IP 地址。

（2）连接成功后，可以实现双向通信，即在服务器上输入字符串，单击发送按钮，可

以把字符串在客户端应用程序界面上显示出来。同时在客户端上输入字符串，单击发送按钮，服务器上也会显示出来。

有了这些需求后，现在笔者就一步一步讲解如何设计这个网络应用程序。

4.5.1　服务器端设计

现在先设计服务器端应用程序，其步骤如下所述。

（1）用 Visual C++应用程序开发向导生成一个对话框程序。单击 OK 按钮，启动 MFC 程序向导，如图 4.7 所示。

（2）在选择 MFC 应用程序类型对话框中，设置应用程序类型为对话框模式（Dialog based），然后单击 Next 按钮，如图 4.8 所示。

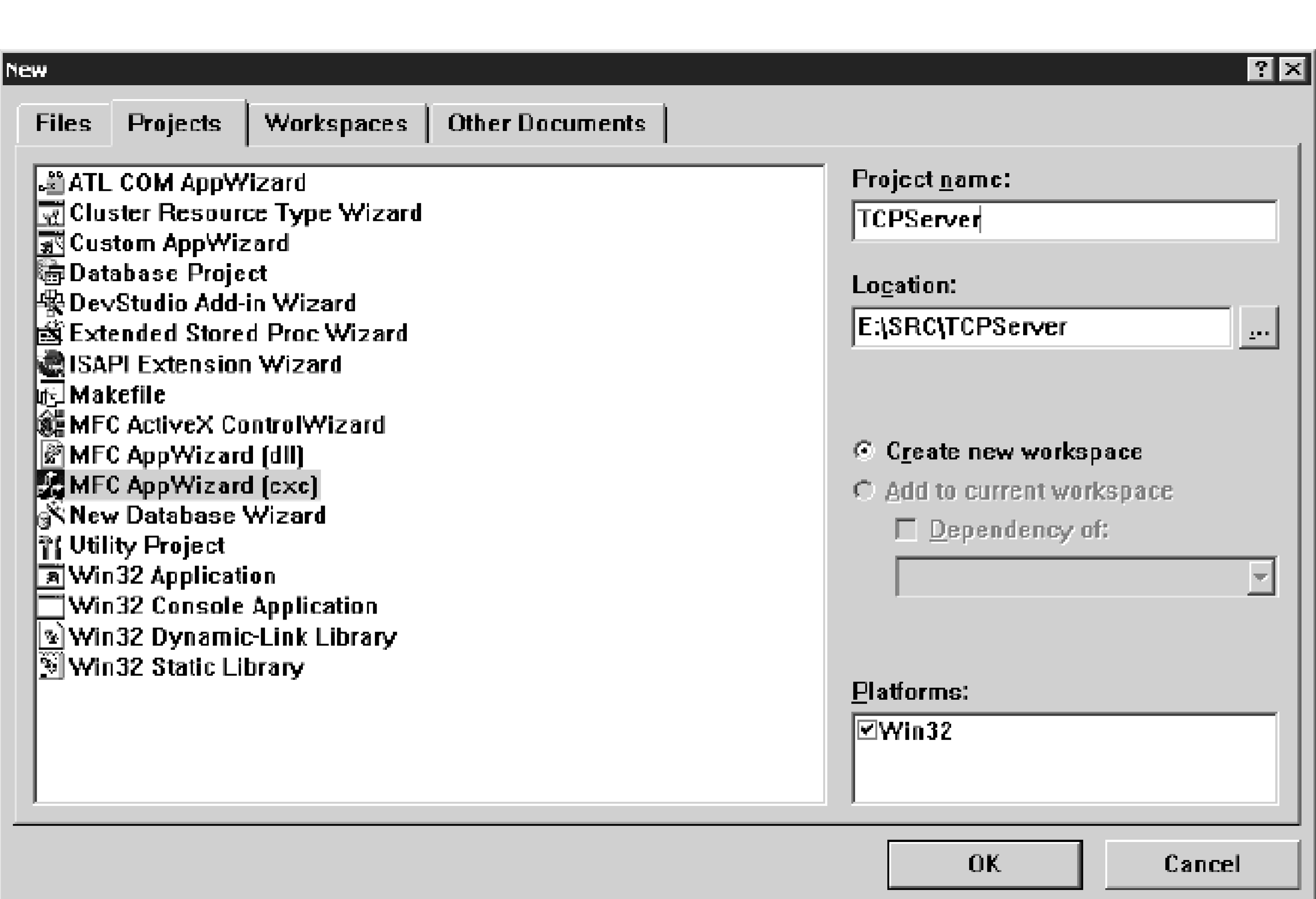


图 4.7　应用程序开发向导

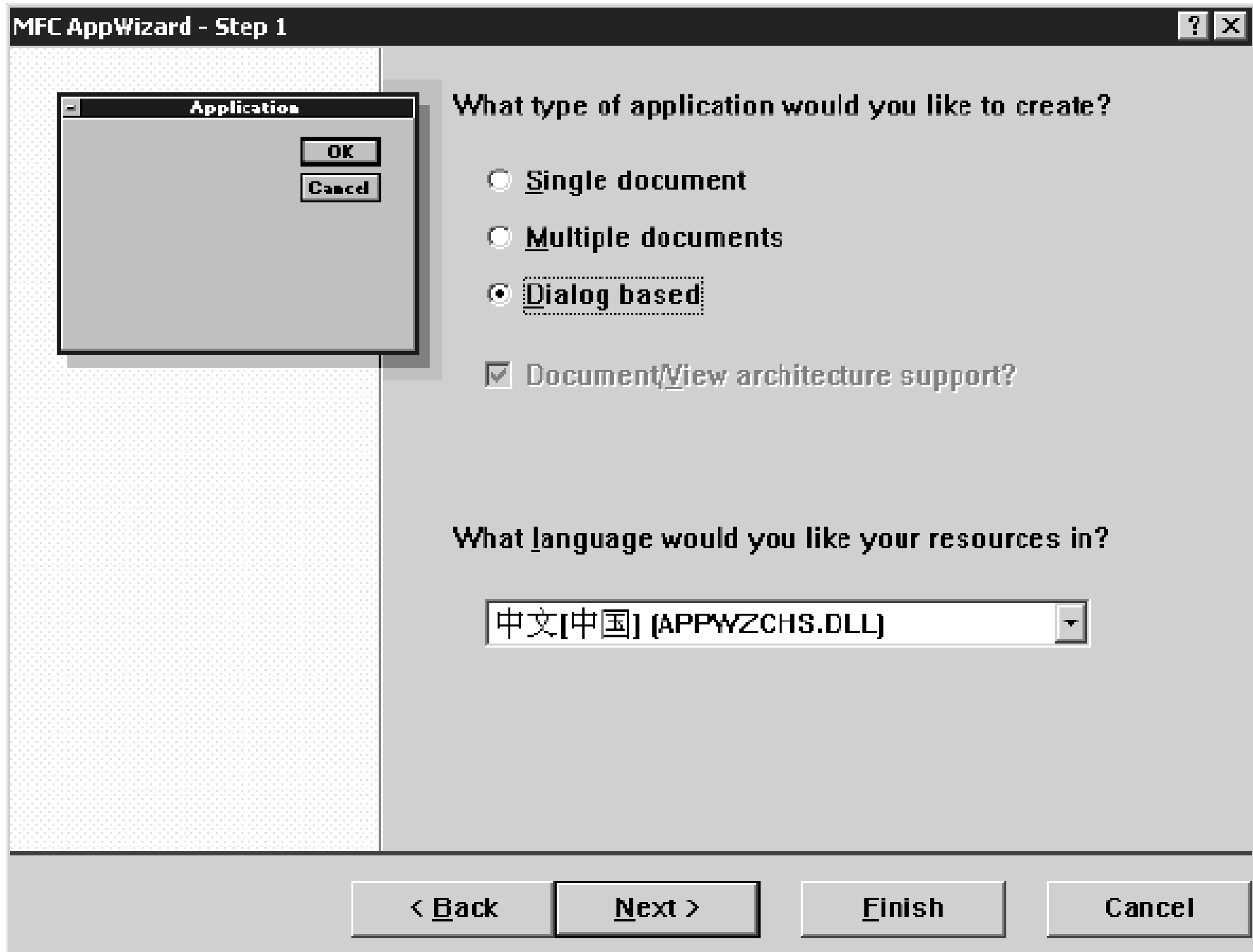


图 4.8　指定当前应用程序模式

（3）在进入的参数设置对话框中选中 Windows Sockets 复选框，然后单击 Next 按钮，如图 4.9 所示。

（4）在进入的确定工程样式对话框中，直接单击 Next 按钮，进入下一步，如图 4.10 所示。

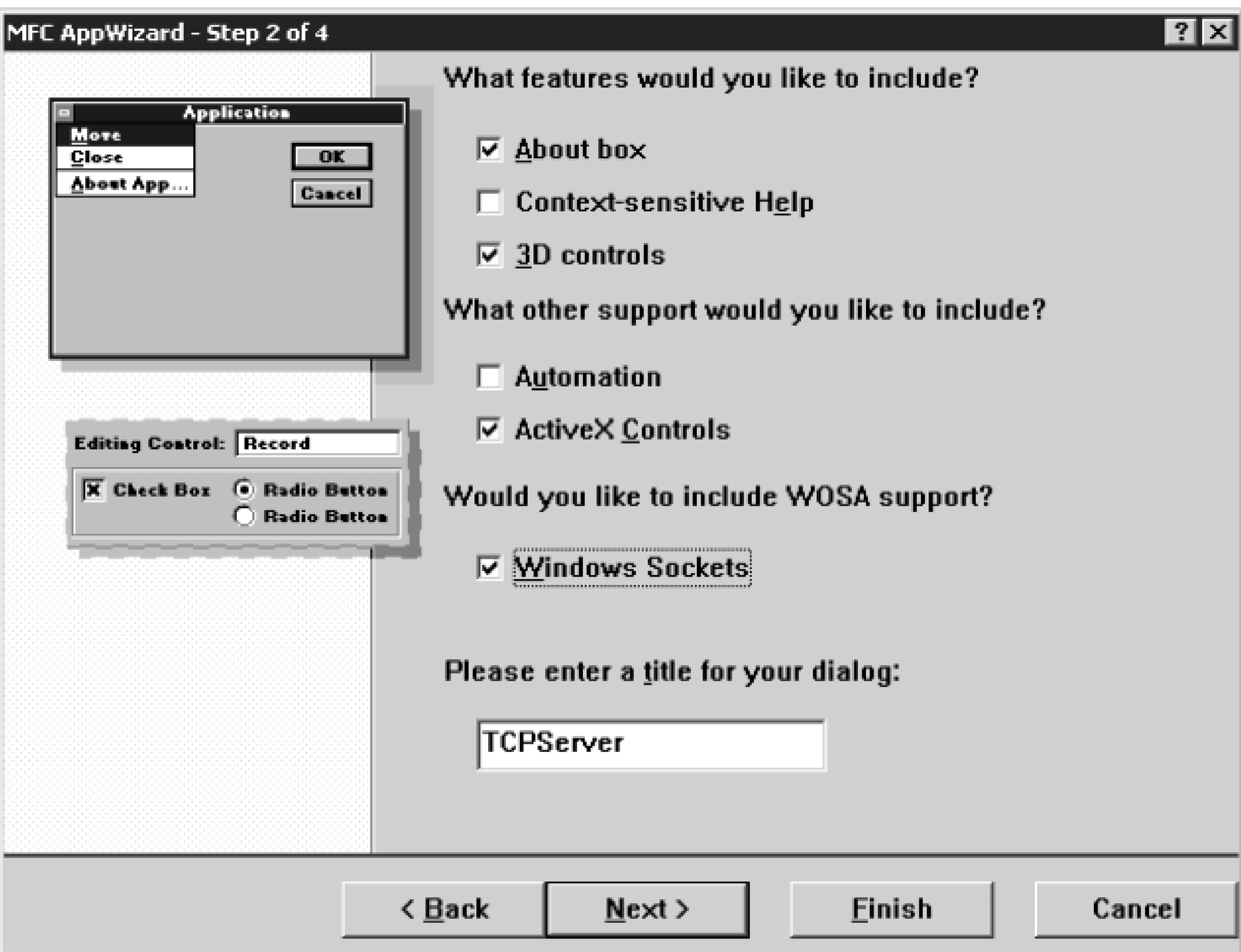


图 4.9　指定当前程序特征

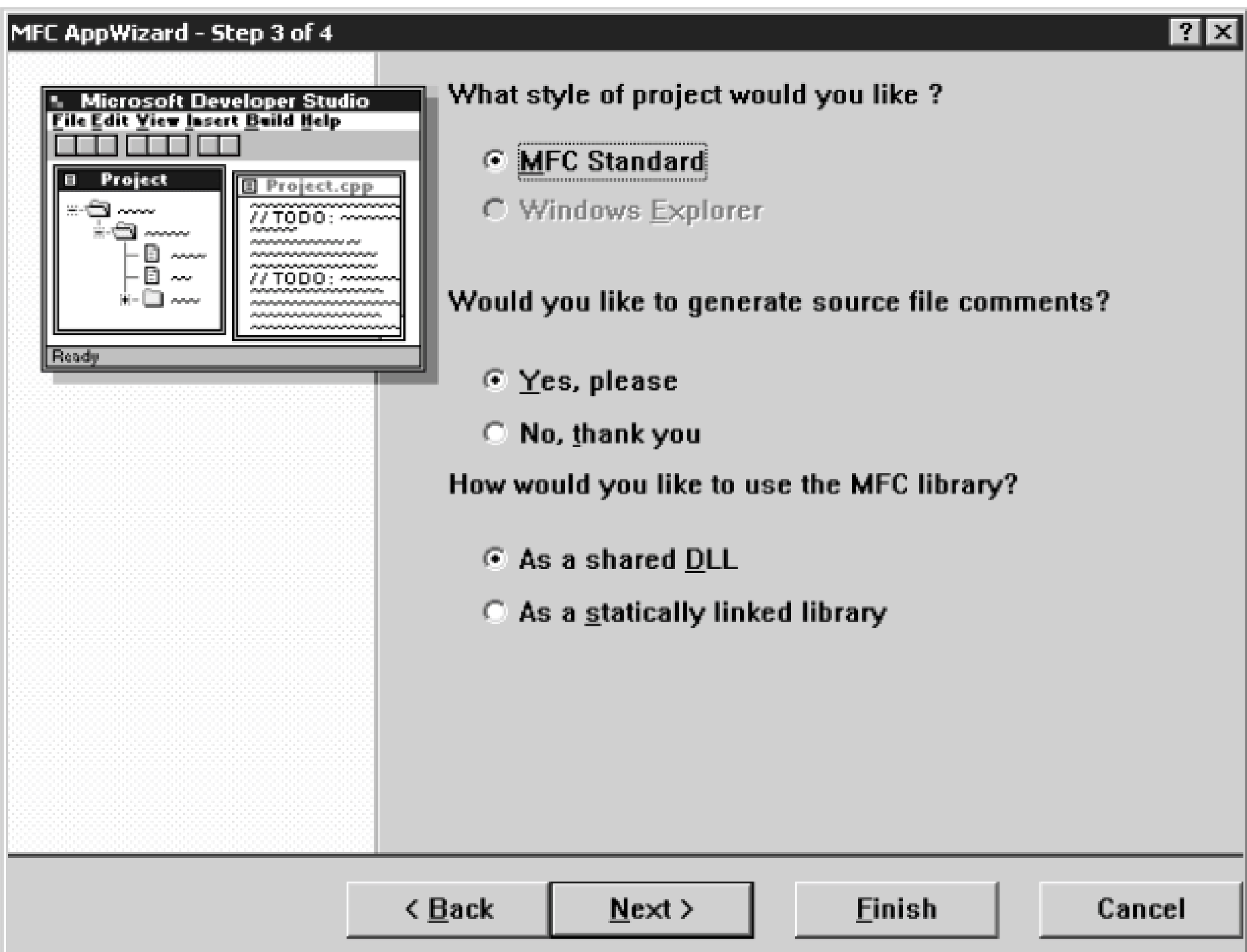


图 4.10　确定工程样式

（5）在类资源对话框中，单击 Finish 按钮，完成对话框程序的设置工作，如图 4.11 所示。

（6）在对话框中加入两个编辑框控件，并按照如图 4.12 所示设计主界面。在主界面设计完毕后，把接收数据编辑框和发送数据编辑框分别与 m_szRecv 和 m_szSend 变量相关联。

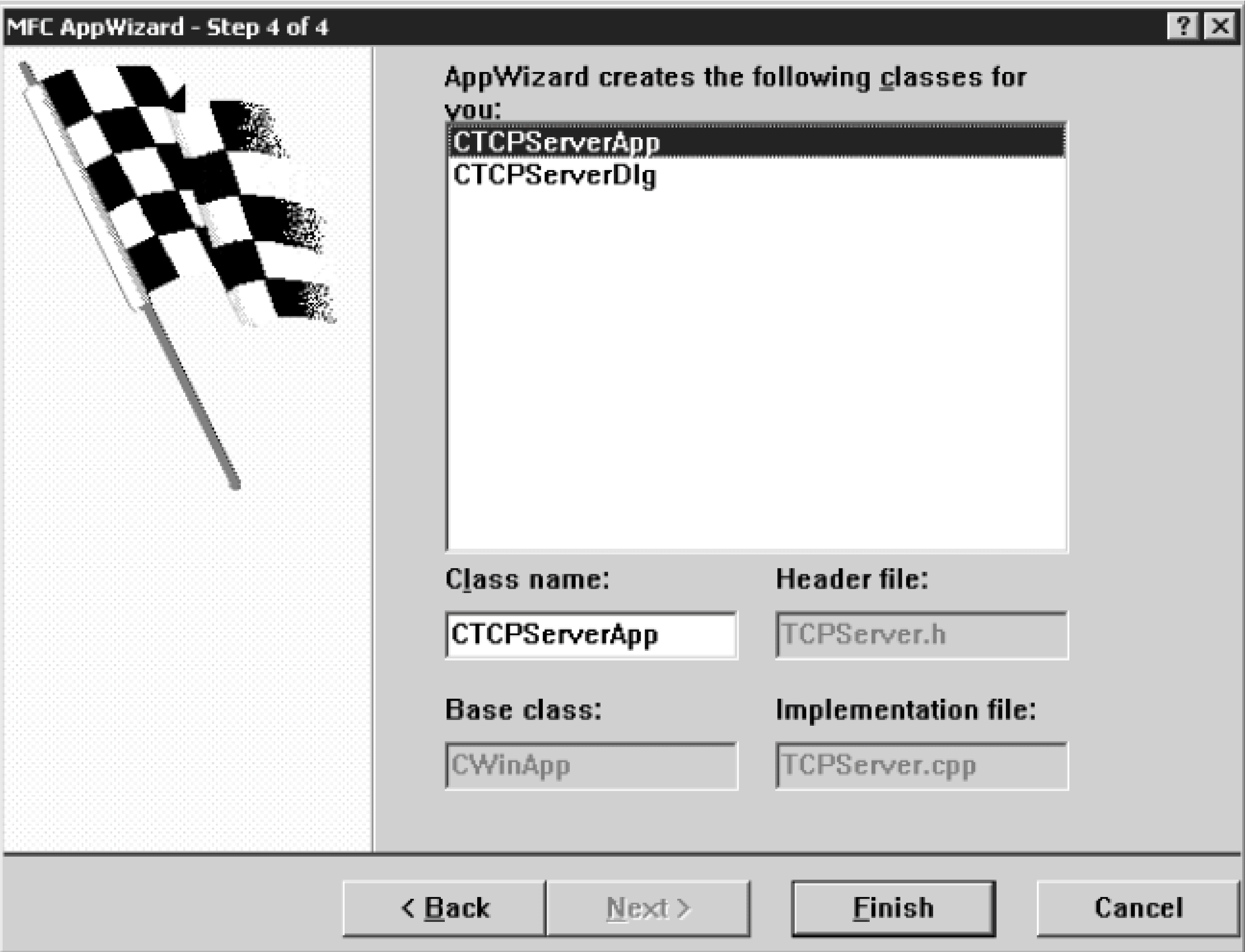


图 4.11　类资源设置

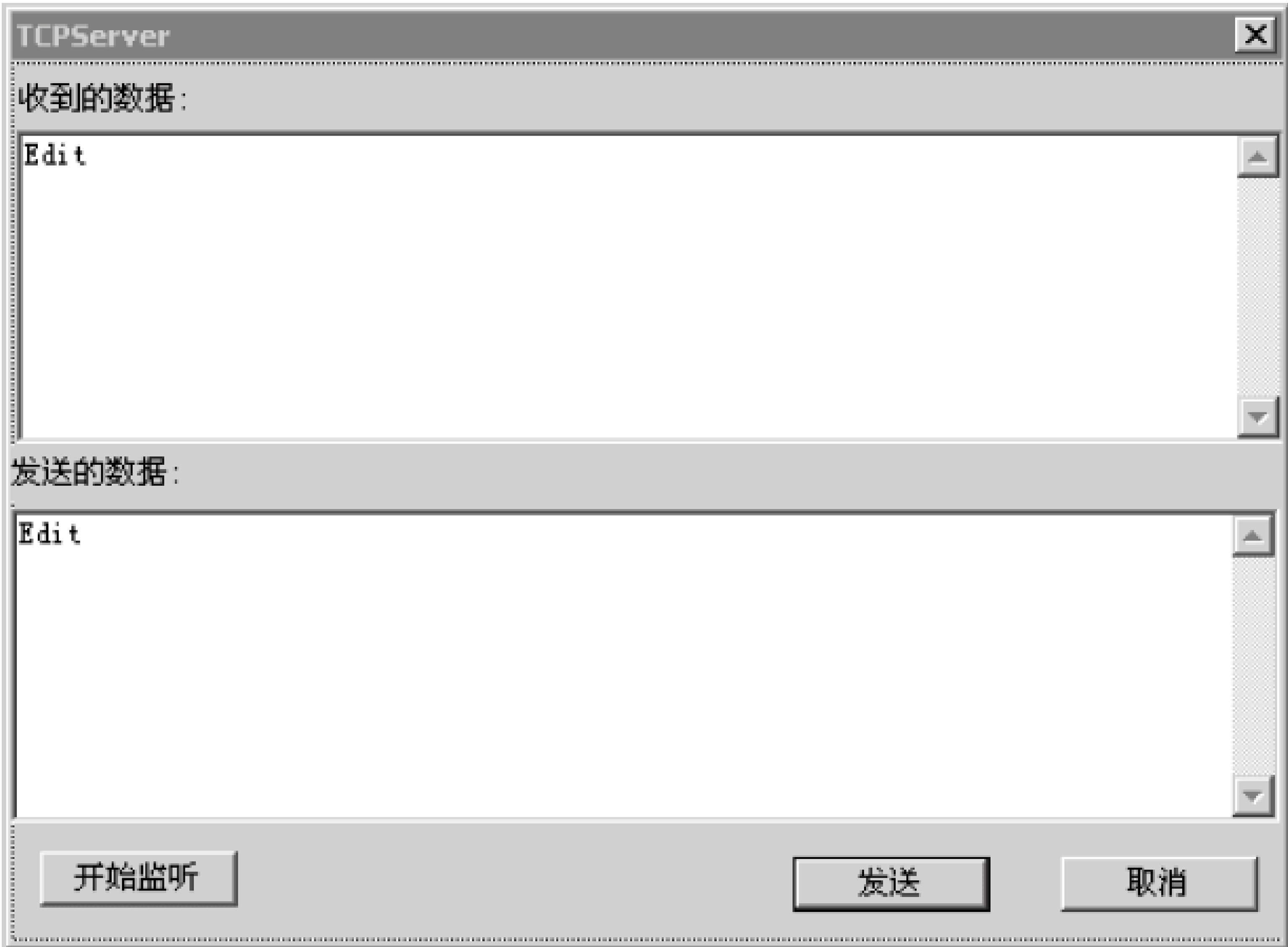


图 4.12　服务器主界面

（7）添加一个新的 CMySocket 类，并设置其继承于 CAsyncSocket 类，然后单击 OK 按钮，如图 4.13 所示。

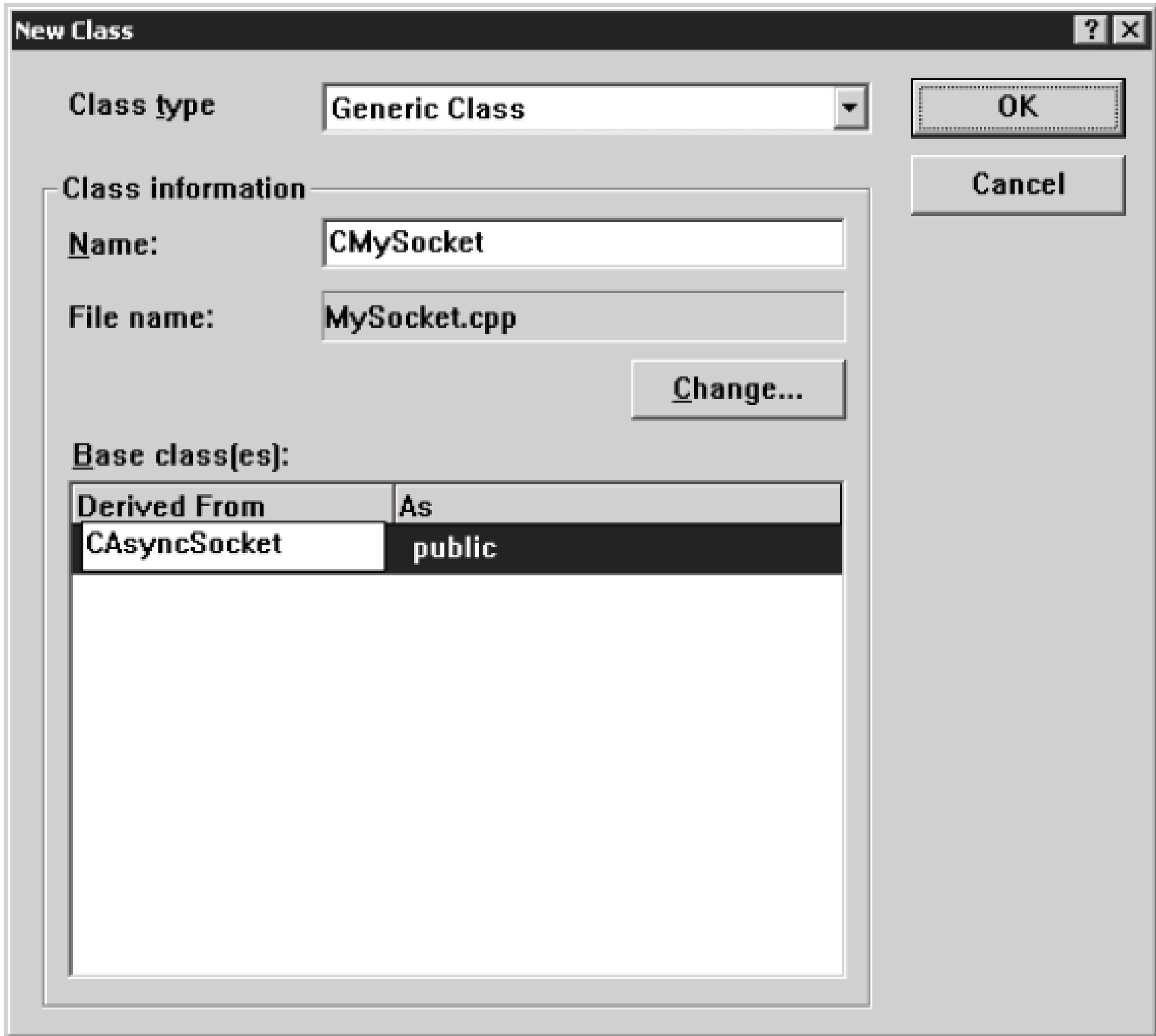


图 4.13　添加新类对话框

（8）按照代码 4.4 所示，把相关代码添加到 CMySocket 类的头文件中。

代码 4.4　CMySocket 类的头文件

```
01  // MySocket.h: interface for the CMySocket class.
02
03  //////////////////////////////////////////////////////////////////////
04
05  #if _MSC_VER > 1000
06  #pragma once
07  #endif // _MSC_VER > 1000
08  class CMySocket : public CAsyncSocket
09  {
10  public:
11      void SetParent(CDialog *pDlg);
12      CMySocket();                                    //构造函数
```

```
13      virtual ~CMySocket();                          //析构函数
14
15  protected:
16      virtual void OnClose(int nErrorCode);          //关闭端口事件响应
17      virtual void OnReceive(int nErrorCode);        //接收数据事件响应
18      virtual void OnConnect(int nErrorCode);        //连接事件响应
19      virtual void OnAccept(int nErrorCode);         //接收事件响应
20
21  private:
22      CDialog *m_pDlg;
23  };
24  #endif
```

（9）按照代码 4.5 所示，把相关代码添加到 CMySocket 类的源文件中。

代码 4.5　CMySocket 类的源文件

```
01  // MySocket.cpp: implementation of the CMySocket class.
02
03  ///////////////////////////////////////////////////////////////////
04  #include "stdafx.h"
05  #include "TCPServer.h"
06  #include "MySocket.h"
07  #include "TCPServerDlg.h"
08
09  #ifdef _DEBUG
10  #undef THIS_FILE
11  static char THIS_FILE[]=__FILE__;
12  #define new DEBUG_NEW
13  #endif
14
15  ////////////////////////////////////////////////////////////////////
16  // 构造和析构函数
17  ////////////////////////////////////////////////////////////////////
18
19  CMySocket::CMySocket()
20  {
21
22  }
23
24  CMySocket::~CMySocket()
25  {
26
27  }
28
29  void CMySocket::SetParent(CDialog *pDlg)    //设置CTCPServerDlg类的指针
30  {
31      m_pDlg=(CTCPServerDlg*)pDlg;
32  }
33
34  void CMySocket::OnAccept(int nErrorCode)    //事件处理函数
35  {
36      if(nErrorCode==0)
37      {//调用 CTCPServerDlg 类中的 OnAccept 成员函数
38          ((CTCPServerDlg*)m_pDlg)->OnAccept();
39      }
40  }
41
42  void CMySocket::OnConnect(int nErrorCode)   //事件处理函数
```

```
43  {
44
45  }
46
47  void CMySocket::OnReceive(int nErrorCode)   //事件处理函数
48  {   //调用 CTCPServerDlg 类中的 OnReceive 成员函数
49      ((CTCPServerDlg*)m_pDlg)->OnReceive();
50  }
51
52  void CMySocket::OnClose(int nErrorCode)     //事件处理函数
53  {   //调用 CTCPServerDlg 类中的 OnClose 成员函数
54      ((CTCPServerDlg*)m_pDlg)->OnClose();
55  }
```

（10）在 CTCPServerDlg 对话框类的头文件中，定义两个 CMySocket 类的对象，如代码 4.6 所示。

代码 4.6　CTCPServerDlg 对话框类的头文件

```
01  // TCPServerDlg.h : header file
02
03
04  #if _MSC_VER > 1000
05  #pragma once
06  #endif // _MSC_VER > 1000
07  #include "MySocket.h"
08  ////////////////////////////////////////////////////////////////////
09  // CTCPServerDlg dialog 对话框
10
11  class CTCPServerDlg : public CDialog
12  {
13
14  public:
15      CTCPServerDlg(CWnd* pParent = NULL);    //CTCPServerDlg 类的构造函数
16      void OnReceive();                       //接收响应函数
17      void OnAccept();                        //连接响应函数
18      void OnClose();                         //关闭响应函数
19      //{{AFX_DATA(CTCPServerDlg)
20      enum { IDD = IDD_TCPSERVER_DIALOG };
21          CString m_szRecv;
22          CString m_szSend;
23      //}}AFX_DATA
24
25      //虚函数声明
26      //{{AFX_VIRTUAL(CTCPServerDlg)
27      protected:
28      virtual void DoDataExchange(CDataExchange* pDX);// DDX/DDV support
29      //}}AFX_VIRTUAL
30
31  //自定义成员声明
32  private:
33      CMySocket m_sockListen;                 //监听端口
34      CMySocket m_sockServer;                 //服务端口
35  protected:
36      HICON m_hIcon;                          //图标句柄
37
38      //Generated message map functions
39      //{{AFX_MSG(CTCPServerDlg)
```

```
40      virtual BOOL OnInitDialog();            //对话框初始化函数
41      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
42      afx_msg void OnPaint();                 //绘画函数
43      afx_msg HCURSOR OnQueryDragIcon();
44      virtual void OnOK();                    //回车按键响应
45      virtual void OnCancel();                //退出按键响应
46      afx_msg void OnSend();                  //发送按钮响应
47      //}}AFX_MSG
48      DECLARE_MESSAGE_MAP()
49  };
50  #endif
```

这里定义两个 CMySocket 类对象的原因是，一个是用于监听连接的套接字，另一个是用于真实通信的套接字。

（11）在 CTCPServerDlg 对话框类的初始化函数中，把当前对话框的指针分别赋值给 m_sockListen 和 m_sockServer。CTCPServerDlg 对话框类的源文件，如代码 4.7 所示。

代码 4.7　CTCPServerDlg 对话框类的源文件

```
01  // TCPServerDlg.cpp 实现源文件
02
03
04  #include "stdafx.h"                 //头文件插入
05  #include "TCPServer.h"
06  #include "TCPServerDlg.h"
07
08
09  //////////////////////////////////////////////////////////////////////
10  // CaboutDlg 关于对话框类
11  class CAboutDlg : public CDialog
12  {
13  public:
14      CAboutDlg();
15
16      enum { IDD = IDD_ABOUTBOX };    //资源名称
17
18      protected:                      //资源加载函数
19      virtual void DoDataExchange(CDataExchange* pDX);
20
21  protected:
22      DECLARE_MESSAGE_MAP()
23  };
24
25  CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
26  {                                   //构造函数
27  }
28  void CAboutDlg::DoDataExchange(CDataExchange* pDX)
29  {                                   //加载资源
30      CDialog::DoDataExchange(pDX);
31  }
32  BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
33  END_MESSAGE_MAP()
34
35  CTCPServerDlg::CTCPServerDlg(CWnd* pParent /*=NULL*/)
36      : CDialog(CTCPServerDlg::IDD, pParent)
37  {                                   //构造函数，加载主图标到标题
38      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
```

```
39      m_szRecv = _T("");                //初始化为空
40      m_szSend = _T("");                //初始化为空
41  }
42  void CTCPServerDlg::DoDataExchange(CDataExchange* pDX)
43  {
44      CDialog::DoDataExchange(pDX);   //加载资源
45      DDX_Text(pDX, IDC_REC_EDIT, m_szRecv); //关联接收控件与变量 m_szRecv
46      DDX_Text(pDX, IDC_SND_EDIT, m_szSend); //关联发送控件与变量 m_szSend
47  }
48  //各种消息响应函数映射
49  BEGIN_MESSAGE_MAP(CTCPServerDlg, CDialog)
50      ON_WM_SYSCOMMAND()
51      ON_WM_PAINT()
52      ON_WM_QUERYDRAGICON()
53      ON_BN_CLICKED(IDC_SEND, OnSend) //当单击发送按钮时，调用 OnSend 函数
54      ON_BN_CLICKED(IDC_ACCEPT_BTN, OnAcceptBtn)
55  END_MESSAGE_MAP()
56
57  BOOL CTCPServerDlg::OnInitDialog()
58  {                                     //初始化函数
59      CDialog::OnInitDialog();
60                                        //加载系统默认菜单
61      CMenu* pSysMenu = GetSystemMenu(FALSE);
62      if (pSysMenu != NULL)
63      {
64          CString strAboutMenu;         //关于菜单
65          strAboutMenu.LoadString(IDS_ABOUTBOX);
66          if (!strAboutMenu.IsEmpty())
67          {
68              pSysMenu->AppendMenu(MF_SEPARATOR);
69              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAbout
                Menu);
70          }
71      }
72
73      SetIcon(m_hIcon, TRUE);           //设置大图标
74      SetIcon(m_hIcon, FALSE);          //设置小图标
75
76      m_sockListen.SetParent(this);     //把当前对话框类的指针设置给对象
77      m_sockServer.SetParent(this);
78      m_sockListen.Create(10000)
79      return TRUE;                      //返回真，表示创建对话框成功
80  }
81
82  void CTCPServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
83  {                                     //系统菜单命令响应
84      if ((nID & 0xFFF0) == IDM_ABOUTBOX)
85      {
86          CAboutDlg dlgAbout;           //创建关于对话框
87          dlgAbout.DoModal();           //弹出关于对话框
88      }
89      else
90      {                                 //调用默认响应
91          CDialog::OnSysCommand(nID, lParam);
92      }
93  }
94
95  void CTCPServerDlg::OnPaint()
```

```
96  {                                   //对话框自绘函数
97      if (IsIconic())
98      {
99          CPaintDC dc(this);          //得到当前设备 DC 上下文
100         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
101         int cxIcon = GetSystemMetrics(SM_CXICON);
102         int cyIcon = GetSystemMetrics(SM_CYICON);
103         CRect rect;
104         GetClientRect(&rect);       //得到当前坐标范围
105         int x = (rect.Width() - cxIcon + 1) / 2;
106         int y = (rect.Height() - cyIcon + 1) / 2;
107         dc.DrawIcon(x, y, m_hIcon); //画图标
108     }
109     else
110     {
111         CDialog::OnPaint();         //调用默认的自绘函数
112     }
113 }
114
115 HCURSOR CTCPServerDlg::OnQueryDragIcon()
116 {
117     return (HCURSOR) m_hIcon;
118 }
119
120 void CTCPServerDlg::OnOK()          //响应回车按键
121 {
122 }
123
124 void CTCPServerDlg::OnCancel()      //响应退出按钮
125 {
126     CDialog::OnCancel();            //调用基类的退出函数
127 }
```

（12）给对话框添加监听响应成员函数，当用户单击“监听”按钮时，就创建端口并开始监听和等待客户端连接。监听按钮的响应函数如代码 4.8 所示。

代码 4.8　监听按钮响应函数

```
01  void CTCPServerDlg::OnAcceptBtn()
02  {
03              m_sockListen.Listen();          //监听端口
04  }
```

（13）添加完监听按钮响应函数后，需要给当前服务器程序添加一个 Accept 响应函数，这个函数是用来响应当有客户端请求连接时，给这个客户端提供一个通信端口。Accept() 函数的实现如代码 4.9 所示。

代码 4.9　CTCPServerDlg 的 Accept()函数实现

```
01  void CTCPServerDlg::OnAccept()
02  {
03      SOCKADDR sockAddr;              //保存地址变量
04      int nSockAddrLen = sizeof(SOCKADDR);
05      CString tmp;                    //临时字符串变量
06      if(m_sockListen.Accept(m_sockServer, &sockAddr, &nSockAddrLen))
07      {                               //接收客户端连接
08          tmp.Format("有客户端连接，来自%d.%d.%d.%d\r\n",
09              (UCHAR)sockAddr.sa_data[2], (UCHAR)sockAddr.sa_data[3],
```

```
10              (UCHAR)sockAddr.sa_data[4], (UCHAR)sockAddr.sa_data[5]);
11          m_szRecv+=tmp;           //更新字符串数据
12          UpdateData(FALSE);       //更新显示
13      }
14  }
```

（14）建立连接成功之后，就需要进行相互通信。所以在这里需要实现 Receive()函数，当套接字上有数据收到时，其就会自动调用这个函数。函数实现如代码 4.10 所示。

代码 4.10　CTCPServerDlg 的 Receive()函数实现

```
01  void CTCPServerDlg::OnReceive()
02  {
03      BYTE byBuf[1024] = {0};                //接收缓冲区
04      int nRecvLen = 0;                      //存储接收到数据的长度
05      nRecvLen = m_sockServer.Receive(byBuf, sizeof(byBuf));
06      CString tmp;                           //临时变量
07      if(nRecvLen > 0)                       //当收到了真实数据
08      {
09          UpdateData();
10          tmp.Format("%s\r\n", byBuf);       //格式化数据
11          m_szRecv+=tmp;                     //更新编辑框变量
12          UpdateData(FALSE);                 //显示
13      }
14      else
15      {                                      //出错提示
16          AfxMessageBox("收到的数据有问题！");
17      }
18  }
```

（15）既然是双向通信，那么有了接收函数自然就会有发送函数。不过这里因为发送是一个主动行为，所以笔者设计为单击“发送”按钮后，把发送文本框中的文本发送出去。发送函数 Send()的实现如代码 4.11 所示。

代码 4.11　CTCPServerDlg 的 Send()函数实现

```
01  void CTCPServerDlg::OnSend()
02  {
03      UpdateData();                     //更新数据到变量
04      int nSendLen = m_sockServer.Send((void*)m_szSend.GetBuffer(0),
05          m_szSend.GetLength());        //发送字符串
06      if(nSendLen > 0)
07      {                                 //大于 0，成功
08          AfxMessageBox("发送成功！");
09      }
10      else
11      {                                 //失败
12          AfxMessageBox("发送失败！");
13      }
14  }
```

（16）如果客户端断开连接，那么服务器端需要把相应生成的新套接字关闭。关闭函数 Close()的实现如代码 4.12 所示。

代码 4.12　CTCPServerDlg 的 Close()函数实现

```
01  void CTCPServerDlg::OnClose()
02  {                                        //收到关闭事件
03      m_sockServer.Close();                //调用关闭函数
04  }
```

4.5.2　客户端设计

设计完服务器端程序后，现在再来设计客户端程序。在这里对于对话框程序创建的向导过程与服务器程序一样，不再复述，只把不同之处列举如下。

（1）在对话框中加入两个编辑框控件，并按照如图4.14所示设计主界面。主界面设计完毕后，把接收数据编辑框和发送数据编辑框分别与 m_szRecv 和 m_szSend 变量相关联。

（2）把服务器端的 CMySocket 源文件修改为如代码 4.13 所示，并加入到当前项目中。

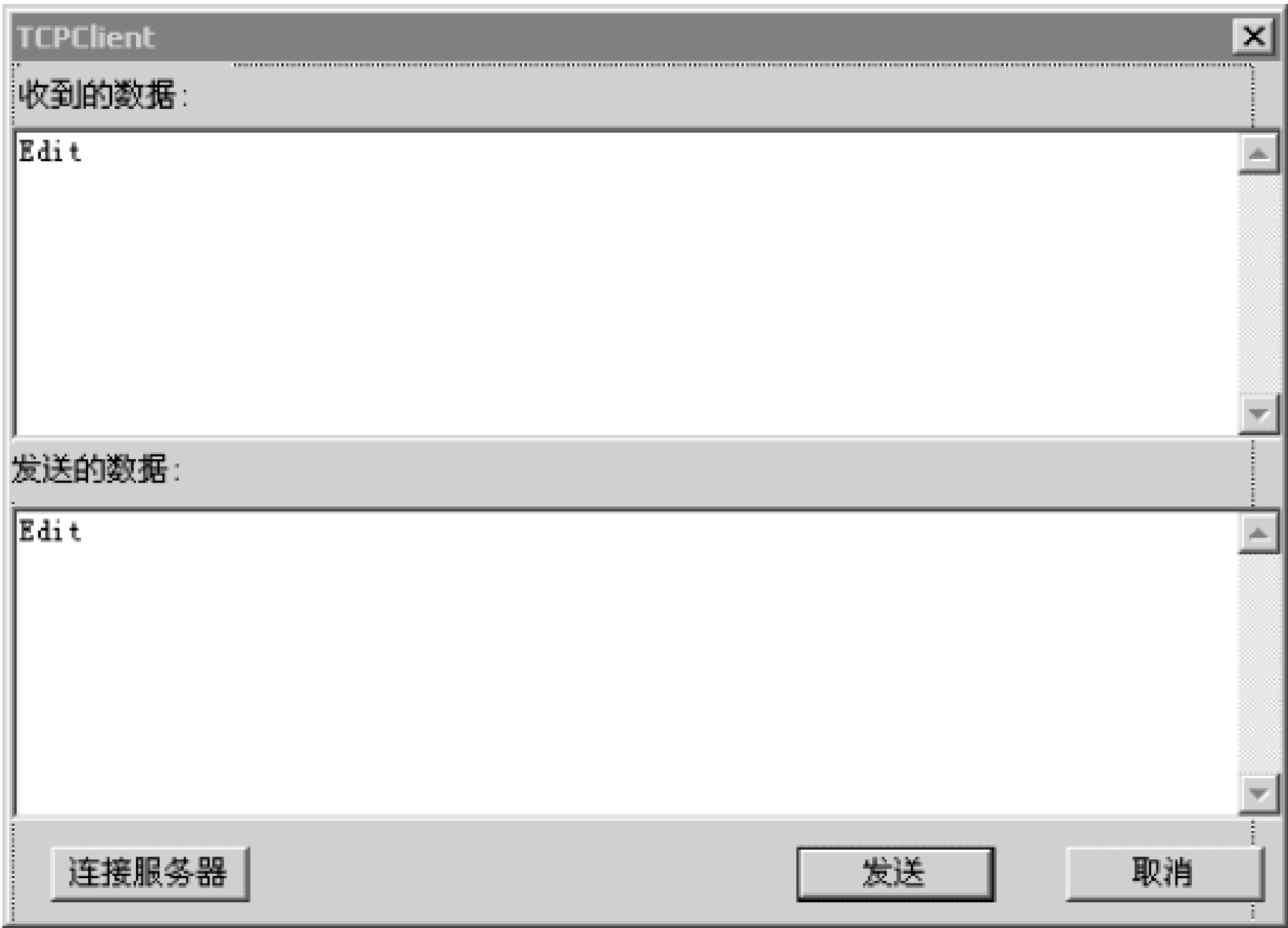


图 4.14　客户端主界面

代码 4.13　客户端 CMySOcket 源文件

```
01  // MySocket.cpp: implementation of the CMySocket class.
02
03  ///////////////////////////////////////////////////////////////////
04
05  #include "stdafx.h"
06  #include "TCPClient.h"
07  #include "MySocket.h"
08  #include "TCPClientDlg.h"
09
10  #ifdef _DEBUG
11  #undef THIS_FILE
12  static char THIS_FILE[]=__FILE__;
13  #define new DEBUG_NEW
14  #endif
15
16  //////////////////////////////////////////////////////////////////
17  // 构造和析构函数
18  ///////////////////////////////////////////////////////////////////
19  CMySocket::CMySocket()                      //构造函数
20  {
21  }
22
23  CMySocket::~CMySocket()                     //析构函数
24  {
25  }
26  void CMySocket::SetParent(CDialog *pDlg)    //设置CTCPClientDlg类的指针
27  {
28      m_pDlg=(CTCPClientDlg*)pDlg;
29  }
30
```

```
31  void CMySocket::OnConnect(int nErrorCode)   //事件处理函数
32  {
33      if(nErrorCode==0)
34      {   //调用 CTCPClientDlg 类中的 OnConnect 成员函数
35          ((CTCPClientDlg*)m_pDlg)->OnConnect();
36  }
37  }
38
39  void CMySocket::OnReceive(int nErrorCode)   //事件处理函数
40  {       //调用 CTCPClientDlg 类中的 OnReceive 成员函数
41      ((CTCPClientDlg*)m_pDlg)->OnReceive();
42  }
43
44  void CMySocket::OnClose(int nErrorCode)     //事件处理函数
45  {       //调用 CTCPClientDlg 类中的 OnClose 成员函数
46      ((CTCPClientDlg*)m_pDlg)->OnClose();
47  }
```

（3）在 CTCPClientDlg 对话框类的头文件中，定义一个 CMySocket 类的对象，代码如代码 4.14 所示。

代码 4.14　CTCPClientDlg 对话框类的头文件

```
01  // TCPClientDlg.h : header file
02
03
04  #if _MSC_VER > 1000
05  #pragma once
06  #endif // _MSC_VER > 1000
07  //////////////////////////////////////////////////////////////////////////
08  // CTCPClientDlg dialog 对话框
09  #include "MySocket.h"
10
11  class CTCPClientDlg : public CDialog
12  {
13  // Construction 成员函数
14  public:
15      void OnConnect();
16      void OnClose();
17      void OnReceive();
18
19      CTCPClientDlg(CWnd* pParent = NULL);//构造函数
20
21  //对话框数据变量
22      //{{AFX_DATA(CTCPClientDlg)
23      enum { IDD = IDD_TCPCLIENT_DIALOG };
24      CString m_szRecv;
25      CString m_szSend;
26      //}}AFX_DATA
27
28      // ClassWizard generated virtual function overrides
29      //{{AFX_VIRTUAL(CTCPClientDlg)
30      protected:
31      virtual void DoDataExchange(CDataExchange* pDX);    //加载资源
32      //}}AFX_VIRTUAL
33  //自定义成员声明
34  private:
35      CMySocket m_sockConnect;                //定义 CMySocket 类对象
```

```
36  protected:
37      HICON m_hIcon;
38      //{{AFX_MSG(CTCPClientDlg)         //函数映射
39      virtual BOOL OnInitDialog();
40      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
41      afx_msg void OnPaint();
42      afx_msg HCURSOR OnQueryDragIcon();
43      afx_msg void OnConnectBtn();
44      afx_msg void OnSend();
45      virtual void OnCancel();
46      //}}AFX_MSG
47      DECLARE_MESSAGE_MAP()
48  };
49  #endif
```

（4）在 CTCPClientDlg 对话框类的初始化函数中，把当前对话框的指针赋值给 m_sockConnect。CTCPClientDlg 对话框类的初始化函数实现，如代码 4.15 所示。

代码 4.15　CTCPClientDlg 对话框类的初始化函数

```
01  BOOL CTCPClientDlg::OnInitDialog()
02  {                                   //初始化函数
03      CDialog::OnInitDialog();
04  
05      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
06      ASSERT(IDM_ABOUTBOX < 0xF000);
07                                      //加载系统默认菜单
08      CMenu* pSysMenu = GetSystemMenu(FALSE);
09      if (pSysMenu != NULL)
10      {
11          CString strAboutMenu;       /*关于菜单*/
12          strAboutMenu.LoadString(IDS_ABOUTBOX);
13          if (!strAboutMenu.IsEmpty())
14          {
15              pSysMenu->AppendMenu(MF_SEPARATOR);
16              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAbout
                Menu);
17          }
18      }
19  
20      SetIcon(m_hIcon, TRUE);             //设置大图标
21      SetIcon(m_hIcon, FALSE);            //设置小图标
22      m_sockConnect.Create();
23      m_sockConnect.SetParent(this);      //把当前对话框类的指针设置给对象
24      return TRUE;                        //返回真，表示创建对话框成功
25  }
```

（5）给对话框添加连接响应成员函数，当用户单击“连接服务器”按钮时，客户端就开始连接服务器。在连接成功后，CMySocket 类会调用 OnConnect 事件响应函数。这两个函数的实现如代码 4.16 所示。

代码 4.16　连接服务器的控制和响应函数

```
01  void CTCPClientDlg::OnConnect()               //连接成功后，自动调用
02  {
03          UpdateData();                         //更新显示数据到变量
04          m_szRecv += CString("连接成功!\r\n");
```

```
05          UpdateData(FALSE);                    //更新显示
06  }
07
08  void CTCPClientDlg::OnConnectBtn()            //连接服务器按钮响应函数
09  {
10      char * lpIP = "127.0.0.1";                //服务器地址
11      m_sockConnect.Connect(lpIP, 10000);       //连接服务器
12  }
```

（6）设计客户端与服务器的发送和接收函数，和服务器的代码差不多，只有接收套接字不同，其实现如代码 4.17 所示。

代码 4.17　客户端的发送与接收函数实现

```
01  void CTCPClientDlg::OnSend()                  //发送函数
02  {
03      UpdateData();                             //更新输入数据到变量
04      int nSendLen = m_sockConnect.Send((void*)m_szSend.GetBuffer(0),
05          m_szSend.GetLength());
06      if(nSendLen > 0)
07      {
08          AfxMessageBox("发送成功!");
09      }
10      else
11      {
12          AfxMessageBox("发送失败!");
13      }
14
15  }

16  void CTCPClientDlg::OnReceive()               //接收函数
17  {
18      BYTE byBuf[1024] = {0};                   //接收缓冲区
19      int nRecvLen = 0;
20      nRecvLen = m_sockConnect.Receive(byBuf, sizeof(byBuf));
21      CString tmp;                              //临时变量
22      if(nRecvLen > 0)
23      {
24          UpdateData();                         //更新数据到变量
25          tmp.Format("%s\r\n", byBuf);          //格式化字符串
26          m_szRecv+=tmp;
27          UpdateData(FALSE);                    //更新显示
28      }
29      else
30      {
31          AfxMessageBox("收到的数据有问题!");
32      }
33  }
```

4.5.3　综合测试

现在就来测试一下服务器与客户端能不能进行通信。测试步骤如下所述。

（1）分别把两个程序代码进行编译（过程参见第 2 章中的 2.5.2 节）。

（2）启动服务器端程序，并单击“开始监听”按钮。

（3）启动客户端程序，单击“连接服务器”按钮。

（4）当服务器收到客户连接时，就可以在发送文本框中输入字符串“欢迎使用本服务器!”，并接收到客户端的“你好!!!”字符串，其效果如图 4.15 所示。

（5）客户端连接服务成功后，当服务器发送字符串，客户端会在接收文本框中把字符串“欢迎使用本服务器!”显示出来，客户端发送“你好!!!”字符串到服务器，如图 4.16 所示。



图 4.15 服务器应用程序执行效果



图 4.16　客户端应用程序执行效果

通过上面的 5 个步骤，表明服务器与客户端已经能够正常通信，已经达到设计的要求。

4.6　总　　结

在这一章中，读者已经学习到 Windows 网络编程的相关基础知识，包括 TCP/IP 协议、OSI 体系、Socket 基本函数及 windows CSockets 类的编程知识。此外，还在本章的最后一节学习了 CAsyncSocket 类的综合应用开发。

现在，各位读者已经了解了 Windows 程序的开发大部分内容，在第 5 章中，笔者将再介绍一下游戏中的多媒体处理的内容。在掌握了这些内容后，就可以真正开始游戏项目的开发了。

在第 5 章中，读者将学习到的内容包括：

（1）Windows 中的图像显示。

（2）Windows 中的声音播放。

（3）各种输入方式的处理。

（4）两个简单的多媒体综合应用开发。

4.7　挑　　战

一、问答题

1．最常用的网络通信协议是什么？OSI 是通用网络协议吗？

2．TCP/IP 协议的特点是什么？由哪些部分组成？

3．TCP 协议与 UDP 协议的区别是什么？

4．什么是 Socket？其通信模式有哪些？

5．MFC 提供的是哪两种网络接口类？其区别是什么？

二、编程题

根据下列要求开发一个 C/S 模式的应用程序系统。

1．设定服务器的服务端口为 8000，采用 TCP 连接方式进行。当有客户端连接时，在收到信息编辑框中提示客户端 IP 地址。

2．连接成功后，可以实现双向通信，即在服务器上输入字符串，单击发送按钮，可以把字符串在客户端应用程序界面上显示出来。同时在客户端上输入字符串，单击发送按钮，服务器上也会显示出来。

3．在客户端界面上增加一个“断开连接”按钮，可以断开与服务器的连接。

4．当客户端断开连接时，服务器给出“客户端已经退出”提示。

第 5 章　游戏中的多媒体处理

为什么在这个世界上，有那么多的人钟情于电脑游戏？究其根本在于其是一种真实的体验，对于大脑来说也是一种全新的感知，对于喜欢接触、挑战新事物、好奇心重的人来说就是一种无法抗拒的魅力。除此以外，电脑游戏受欢迎的最主要的原因就是其有强大的多媒体处理能力。能够给人从视觉、听觉、触觉等多个感觉器官直接的感受。本章就主要介绍游戏中的多媒体的概念及其处理。

本章主要涉及的内容如下：

- 多媒体的概念及特点：知道什么是多媒体及其特点是什么。
- 游戏中的图像：掌握 Windows 中各种类型的图像，以及在 Visual C++中如何显示图像。
- 游戏中的声音：掌握 Windows 中各种类型的声音，以及在 Visual C++中的声音是如何播放的。
- 两个开发实例：学习设计一个简单的 MP3 播放器及图像浏览器。

5.1　游戏的多媒体

探索新事物是人类的本性，电脑游戏能代替很多无法从事的危险活动，如极限运动，特警等。游戏开发由一些极其聪明的人来设计逻辑并将其量化，然后由美工画出一个个美轮美奂、极其逼真的图形，最后由程序员进行最终的整合。由于整个过程是完全的大脑产物，所以人类疯狂的大脑在这里得到了最忠实的展现：没有做不到只有想不到，任何模拟现实、超越现实的作品都被不断地开发出来。电脑游戏具有的这些魅力就是多媒体的功劳。

5.1.1　多媒体的概念

说了这么多的多媒体（Multimedia），那么到底什么是多媒体呢？其实多媒体的概念比较广泛。

传统意义上讲，媒体（Media）就是实现信息交流的中介物。简单点说，就是各种信息的载体，也称为媒介。多媒体就是多重（Multi）媒体的意思，可以理解为直接作用于人的感官的文字、图形、图像、动画、声音和视频等各种媒体的统称，即多种信息载体的表现和传递方式。包括电视、电影、VCD、DVD、电脑、网络等。

而现代的解释为：信息的正文、图形、声音、图像、动画，被称为媒体。从电脑处理信息的角度来看，可以将自然界和人类社会原始信息的数据、文字、有声的语言、音乐、绘画、动画、图像（静态的照片和动态的电影、电视和录像）等，归结为 3 种最基本的媒

体：声、图、文。

旧式的计算机只能够处理单一媒体“文”。电视能够传播声、图、文集成信息，但其并不是多媒体系统。通过电视，只能单向被动地接收信息，不能双向地、主动地处理信息，没有所谓的交互性。可视电话虽然有交互性，但仅仅能够听到声音，见到谈话人的形象，也不是多媒体。

所谓多媒体，是指能够同时采集、处理、编辑、存储和展示两个或以上不同类型信息媒体的技术，这些信息媒体包括文字、声音、图形、图像、动画和活动影像等。

多媒体是能够依靠数字技术实现数字控制和数字媒体的汇合体。在这之中电脑是数字控制系统，而数字媒体是当今音频和视频最先进的存储和传播形式。事实上可以简单地认为多媒体就是电脑和电视的结合。电脑的能力达到实时处理电视和声音数据流的水平，这时多媒体就诞生了。

说明：现代多媒体可以通过控制实现真实的交互，让电脑按使用者的需要去做。

5.1.2　多媒体技术的特点

多媒体技术有以下主要特点。

- 集成性：能够对多个种类的信息进行统一获取、存储、组织与合成能力。
- 控制性：多媒体技术是以电脑为中心，综合处理和控制多媒体信息，并按人的要求以多种媒体的形式表现出来，同时作用于人的多种感觉器官。
- 非线性：改变人们传统循序性的读写模式。以往人们读写方式大都采用章、节、页的框架，循序渐进地获取知识，而多媒体技术将借助超文本链接的方法，把内容以一种更灵活、更具变化的方式呈现给读者。例如，网页上的各种链接。
- 交互性：是多媒体应用有别于传统信息交流媒体的主要特点之一。传统信息交流媒体只能单向地、被动地传播信息，如电视，报纸和书籍等。而多媒体技术则可以实现人对信息的主动选择和控制，如网站浏览、视频点播等。
- 实时性：当用户给出操作命令时，相应的多媒体信息都能够得到实时控制。
- 信息使用的方便性：用户可以按照自己的需要、兴趣、任务要求、偏爱和认知特点来使用信息，任取图、文、声等信息表现形式。
- 信息结构的动态性：用户可以按照自己的目的和认知特征重新组织信息，增加、删除或修改结点，重新建立链来更新信息内容。

5.1.3　多媒体能做什么

读者肯定会有疑问，多媒体到底能做什么呢？其实多媒体的主要功能是其能展示信息、交流思想和抒发情感，能让你看到、听到和理解其他人的思想。换句话说，多媒体是一种新的通信方式。

多媒体的主要作用很广泛，笔者在这里只列举其中的几种。

- 视听享受：播放电影、欣赏音乐。
- 互动操作：玩电脑游戏、上网冲浪。

- 文字办公：处理文件、练习打字、图片展示及修订。
- 辅助教学：制作幻灯片、动画片及课件。
- 广告效果：户外广告、电视广告等。

5.2　认识各种多媒体文件

多媒体文件是指电脑中以文件的形式存储的各种不同编码的数据，其是二进制数据的集合。文件的命名遵循特定的系统规则，在 Windows 系统上一般由主名和扩展名两部分组成，主名与扩展名之间用“.”隔开，扩展名用于表示文件的格式类型。

多媒体文件的类型包括如下几种。

- 文本类型：文本是以文字和各种专用符号表达的信息形式，其是现实生活中使用得最多的一种信息存储和传递方式。用文本表达信息给人充分的想象空间，主要用于对知识的描述性表示，如阐述概念、定义、原理和问题，以及显示标题、菜单等内容。
- 图像类型：图像是多媒体软件中最重要的信息表现形式之一，是决定一个多媒体软件视觉效果的关键因素。
- 动画类型：动画是利用人的视觉暂留特性，快速播放一系列连续运动变化的图形图像，也包括画面的缩放、旋转、变换、淡入淡出等特殊效果。通过动画可以把抽象的内容形象化，使许多难以理解的内容变得生动有趣。合理使用动画可以达到事半功倍的效果。
- 声音类型：声音是人们用来传递信息、交流感情最方便、最熟悉的方式之一。在 Windows 系统中，按其表达形式，可将声音分为声音和音乐两种。
- 视频影像：视频影像具有时序性与丰富的信息内涵，常用于交待事物的发展过程。例如，电影和电视，有声有色，在多媒体中充当起重要的角色。

5.2.1　Windows 中的文本文件

下面通过文本文件的扩展名，来逐一介绍当前常见的文本文件格式。

- TXT 格式：文本文档，内容都是比较简单的文字消息。
- DOC 格式：Microsoft Word 文档，内容包括文字、图像及声音等。
- XLS 格式：Microsoft Excel 工作表，内容包括简单的数据和表格。
- ASP 格式：活动服务器文档。
- COL 格式：HTML 帮助文件。
- DIC 格式：文本文档。
- HLP 格式：帮助类文本文件。
- INF 格式：安装信息类文本文件。
- INI 格式：配置类文本文件。
- LOG 格式：日志类文本文档。

5.2.2　Windows 中的图像文件

一般来说，目前的图形（图像）格式大致可以分为两大类：一类为位图；另一类称为描绘类、矢量类或面向对象的图形（图像）。前者是以点阵形式描述图形（图像），后者是以数学方法描述的一种由几何元素组成的图形（图像）。一般来说，后者对图像的表达细致、真实，缩放后图形（图像）的分辨率不变，在专业级的图形（图像）处理中运用较多。

在介绍图形（图像）格式前，笔者觉得实在有必要先了解一下图形（图像）的一些相关技术指标：分辨率、色彩数、图形灰度。

- 分辨率：分为屏幕分辨率和输出分辨率两种，前者用每英寸行数表示，数值越大图形（图像）质量越好；后者衡量输出设备的精度，以每英寸的像素点数表示。
- 色彩数和图形灰度：用位（bit）表示，一般写成 2 的 n 次方，n 代表位数。当图形（图像）达到 24 位时，可表现 1677 万种颜色，即真彩。灰度的表示法类似。

下面就通过图形文件的扩展名（就是如图 1.bmp 这样的），来逐一认识当前常见的图形文件格式。

- BMP 格式：是 Windows 操作系统中的标准图像文件格式，能够被多种 Windows 应用程序所支持。这种格式的特点是包含的图像信息较丰富，几乎不进行压缩，占用磁盘空间过大。
- GIF 格式：GIF 格式的特点是压缩比高，磁盘空间占用较少，目前 Internet 上大量采用的彩色动画文件多为这种格式的文件。GIF 有个小小的缺点，即不能存储超过 256 色的图像。
- JPEG 格式：JPEG 文件的扩展名为.jpg 或.jpeg，可以用最少的磁盘空间得到较好的图像质量。因为 JPEG 格式的文件尺寸较小，下载速度快，使得 Web 页有可能以较短的下载时间提供大量美观的图像。
- JPEG2000 格式：与 JPEG 相比，具备更高压缩率以及更多新功能的新一代静态影像压缩技术。JPEG2000 可应用于传统的 JPEG 市场，如扫描仪、数码相机等，亦可应用于新兴领域，如网络传输、无线通讯等。
- TIFF 格式：特点是图像格式复杂、存贮信息多。该格式有压缩和非压缩两种形式。
- DIB 格式：描述图像的能力基本与 BMP 相同，并且能运行于多种硬件平台，只是文件较大。
- PCP 格式：由 ZSoft 公司创建的一种经过压缩且节约磁盘空间的 PC 位图格式，其最高可表现 24 位图形（图像）。过去有一定的市场，但随着 JPEG 的兴起，其地位已逐渐日落西山。
- PSD 格式：是著名的 Adobe 公司的图像处理软件 Photoshop 的专用格式 Photoshop Document（PSD）。PSD 其实是 Photoshop 进行平面设计的一张“草稿图”，里面包含有各种图层、通道、遮罩等多种设计的样稿，以便于下次打开文件时可以修改上一次的设计。在 Photoshop 所支持的各种图像格式中，PSD 的存取速度比其他格式快很多，功能也很强大。
- PNG 格式：PNG（PortableNetworkGraphics）是一种新兴的网络图像格式。PNG 是目前保证最不失真的格式，其吸取了 GIF 和 JPG 二者的优点，存贮形式丰富，

兼有 GIF 和 JPG 的色彩模式；其另一个特点是能把图像文件压缩到极限以利于网络传输，但又能保留所有与图像品质有关的信息，PNG 采用无损压缩方式来减少文件的大小。

- DIF：AutoCAD 中的图形文件，其以 ASCII 方式存储图形，表现图形在尺寸大小方面十分精确，可以被 CorelDraw、3DS 等大型软件调用编辑。
- WMF：Microsoft Windows 图元文件，具有文件短小、图案造型化的特点。该类图形比较粗糙，并只能在 Microsoft Office 中调用编辑。
- CDR：CorelDraw 的文件格式。另外，CDX 是所有 CorelDraw 应用程序均能使用的图形（图像）文件，是发展成熟的 CDR 文件。
- IFF：用于大型超级图形处理平台，比如 AMIGA 机，好莱坞的特技大片多采用该图形格式处理。图形（图像）效果，包括色彩纹理等逼真再现原景。当然，该格式耗用的内存外存等计算机资源也十分巨大。
- TGA：是 True vision 公司为其显示卡开发的图形文件格式，创建时期较早，最高色彩数可达 32 位。VDA、PIX、WIN、BPX、ICB 等均属其旁系。
- PCD（Photo CD）：由 KODAK 公司开发，其他软件系统对其只能读取。

5.2.3　Windows 中的声音文件

接下来介绍几种目前最流行的多媒体声音文件效果，让读者认识一下，如下所示。

- WAV 格式：Microsoft 公司的音频文件格式，其来源于对声音模拟波形的采样。用不同的采样频率对声音的模拟波形进行采样可以得到一系列离散的采样点，以不同的量化位数（8 位或 16 位）把这些采样点的值转换成二进制数，然后存入磁盘，这就产生了声音的 WAV 文件，即波形文件。Microsoft Sound System 软件 Sound Finder 可以转换 AIF SND 和 VOD 文件到 WAV 格式。利用该格式记录的声音文件能够和原声基本一致，质量非常高，但这样做的代价就是文件太大。
- MOD 格式：该格式的文件里存放乐谱和乐曲使用的各种音色样本，具有回放效果明确、音色种类无限等优点。但其也有一些致命弱点，以至于现在已经逐渐被淘汰，目前只有 MOD 迷及一些游戏程序中尚在使用。
- MP3 格式：现在最流行的声音文件格式，因其压缩率大（采用破坏压缩），在网络可视电话通信方面应用广泛，但和 CD 唱片相比，音质不能令人非常满意。
- RA 格式：这种格式真可谓是网络的灵魂，强大的压缩量和极小的失真使其在众多格式中脱颖而出。和 MP3 相同，它是为了解决网络传输带宽资源而设计的，因此主要目标是压缩比和容错性，其次才是音质。
- CMF 格式：该格式是 Creative 公司的专用音乐格式，和 MIDI 差不多，只是音色、效果上有些特色，专用于 FM 声卡，但其兼容性也很差。
- CDA 格式：是唱片上采用的格式，又叫“红皮书”格式，记录的是波形流，绝对纯正、HIFI。但缺点是无法编辑，文件长度太大。
- MID 格式：目前最成熟的音乐格式，是由世界上主要电子乐器制造厂商建立起来的一个通信标准，以规定计算机音乐程序电子合成器和其他电子设备之间交换信息与控制信号的方法。MIDI 文件中包含音符定时和多达 16 个通道的乐器定义，

每个音符包括键通道号持续时间音量和统一的标准格式，能够模仿原始乐器的各种演奏技巧甚至无法演奏的效果。由于 MIDI 文件记录的不是乐曲本身，而是一些描述乐曲演奏过程中的指令，所以文件的长度非常小。

- WMA 格式：是微软公司推出的与 MP3 格式齐名的一种新的音频格式。由于 WMA 在压缩比和音质方面都超过了 MP3，更是远胜于 RA，即使在较低的采样频率下也能产生较好的音质，再加上 WMA 有微软的 Windows Media Player 做其强大的后盾，所以一经推出就赢得一片喝彩。网上的许多音乐纷纷转向 WMA，许多播放器软件也纷纷开发出支持 WMA 格式的插件程序。或许用不了多长时间，WMA 就会成为网络音频的主要格式。
- VOC 格式：Creative 公司波形音频文件格式，也是声霸卡（sound blaster）使用的音频文件格式。每个 VOC 文件由文件头块（header block）和音频数据块（data block）组成。文件头包含一个标识版本号和一个指向数据块起始的指针。数据块分成各种类型的子块。如声音数据静音标识 ASCII 码文件重复的结果重复，以及终止标志、扩展块等。
- RMI 格式：微软公司的 MIDI 文件格式，其可以包括图片标记和文本。
- PCM 格式：模拟音频信号经模数转换（A/D 变换）直接形成的二进制序列，该文件没有附加的文件头和文件结束标志。Windows 的 Convert 工具可以把 PCM 音频格式的文件转换成 Microsoft 的 WAV 格式的文件。
- AIF 格式：Apple 计算机的音频文件格式。Windows 的 Convert 工具同样可以把 AIF 格式的文件换成 Microsoft 的 WAV 格式的文件。

5.2.4　Windows 中的视频文件

视频文件在电脑中存放的格式有很多，而且有一些格式，其扩展名是相同的，所以这里不再以扩展名来区分。目前最流行的几种格式如下所述。

- MPEG 格式：扩展名为 mpg、dat 等。其是运动图像专家组格式，是各种 MPEG 视频格式的总称，VCD（MPEG-1）、SVCD（MPEG-2）、DVD（MPEG-2）就是这种格式。MPEG 格式是运动图像压缩算法的国际标准，采用了有损压缩方法从而减少运动图像中的冗余信息。MPEG 的压缩方法保留相邻两幅画面绝大多数相同的部分，而把后续图像中和前面图像有冗余的部分去除，从而达到压缩的目的。
- DivX/XviD 格式：扩展名为 avi。其中 DivX 是由 MPEG-4 衍生出的一种视频编码压缩标准，即通常所说的 DVDrip 格式，采用了 MPEG4 的压缩算法，同时又综合了 MPEG-4 与 MP3 各方面的技术，说白了就是使用 DivX 压缩技术对 DVD 盘片的视频图像进行高质量压缩，同时用 MP3 或 AC3 对音频进行压缩，然后再将视频与音频合成，并加上相应的外挂字幕文件而形成的视频格式。其画质直逼 DVD，而体积只有 DVD 的数分之一。XviD 与 DivX 几乎相同，是开源的 DivX，不收费，而使用 DivX 要收费。
- MOV 格式：扩展名为 mov。是苹果公司创立的一种视频格式，其采用面向最终用户桌面系统的低成本、全运动视频的方式，在软件压缩和解压缩中也开始采用这种方式。其向量量化是 Quicktime 软件的压缩技术之一，在最高为 30 帧/秒下提供的

视频分辨率是 320×240，其压缩率能从 25 到 200。在很长的一段时间里，这种格式文件都是只在苹果公司的 MAC 机上存在，后来才发展到支持 Windows 平台。

- AVI 格式：扩展名为 avi。是微软公司采用的音频视频交错格式，也是一种桌面系统上的低成本、低分辨率的视频格式。AVI 可在 160×120 的视窗中以 15 帧/秒回放视频，并可带有 8 位的声音，也可以在 VGA 或超级 VGA 监视器上回放。AVI 很重要的一个特点是可伸缩性，使用 AVI 算法时的性能依赖于与其一起使用的基础硬件。
- WMV 格式：扩展名为 wmv。是微软推出的一种采用独立编码方式并且可以直接在网上实时观看视频节目的文件压缩格式。WMV 格式的主要优点包括：本地或网络回放、可扩充的媒体类型、可伸缩的媒体类型、多语言支持、环境独立性、丰富的流间关系，以及扩展性等。
- REAL 格式：扩展名为 rm、ram，是 Real Networks 公司所制定的音频视频压缩规范，称之为 Real Media，Real Media 可以根据不同的网络传输速率制定出不同的压缩比率，从而实现在低速率的网络上进行影像数据实时传送和播放。这种格式的另一个特点是用户使用 RealPlayer 或 RealOne Player 播放器可以在不下载音频/视频内容的条件下实现在线播放。
- RMVB 格式：扩展名为 rmvb、rm。其是一种由 RM 视频格式升级延伸出的新视频格式。先进之处在于打破了原先 RM 格式那种平均压缩采样的方式，在保证平均压缩比的基础上合理利用比特率资源，就是说静止和动作场面少的画面场景采用较低的编码速率，这样可以留出更多的带宽空间，而这些带宽会在出现快速运动的画面场景时被利用。这样在保证了静止画面质量的前提下，大幅地提高了运动图像的画面质量，从而图像质量和文件大小之间就达到了微妙的平衡。

另外，相对于 DVDrip 格式，RMVB 视频也有着较明显的优势，一部大小为 700MB 左右的 DVD 影片，如果将其转录成同样视听品质的 RMVB 格式，其大小最多也就 400MB 左右。不仅如此，这种视频格式还具有内置字幕和无需外挂插件支持等独特优点。

- ASF 格式：扩展名为 asf，是微软公司为了和 Real 的竞争而发展出来的一种可以直接在网上观看视频节目的文件压缩格式。其使用了 MPEG4 的压缩算法，所以压缩率和图像的质量都很不错。ASF 的图像质量比 VCD 差一点，但比同是视频“流”格式的 RAM 格式要好。
- DV-AVI 格式：扩展名为 avi。是由索尼、松下、JVC 等多家厂商联合推出的一种家用数字视频格式。目前非常流行的数码摄像机就是使用这种格式记录视频数据的。可以通过电脑的 IEEE 1394 端口传输视频数据到电脑，也可以将电脑中编辑好的视频数据回录到数码摄像机中。这种视频格式的文件扩展名一般也是.avi，所以习惯地称为 DV-AVI 格式。
- H.261：扩展名为 3gp。又称为 P*64，其中 P 为 64kb/s 的取值范围，是 1～30 的可变参数，最初是针对在 ISDN 上实现电信会议应用，特别是面对面的可视电话和视频会议而设计的。实际的编码算法类似于 MPEG 算法，但不能与后者兼容。H.261 在实时编码时比 MPEG 所占用的 CPU 运算量少得多，此算法为了优化带宽占用量，引进了在图像质量与运动幅度之间的平衡折中机制。也就是说，剧烈运动的图像比相对静止的图像质量要差。因此这种方法是属于恒定码流可变质量编码而非恒

定质量可变码流编码。

- H.263/H.263+格式：扩展名为 3gp。H.263 是国际电联 ITU-T 的一个标准草案，是为低码流通信而设计的。但实际上这个标准可用在很宽的码流范围，而非只用于低码流应用，在许多应用中已经取代了 H.261。H.263 的编码算法与 H.261 一样，但做了一些改善和改变，以提高性能和纠错能力。H.263 标准在低码率下能够提供比 H.261 更好的图像效果。

5.3　游戏中图像的显示

不知道读者朋友们有没有玩过“星际争霸”这款游戏。其无论从游戏设计还是游戏画面上都非常出色。

一款游戏效果如何，主要由玩家所看到的显示效果来决定。一款游戏就算设计得再好，如果没有一个很好的图像显示效果，那展现在玩家眼前的东西将大打折扣。所以，游戏中图像的显示，是游戏开发中的重中之重。笔者将在本节中详细讲解 Visual C++中的 3 种图像显示方法。

5.3.1　使用 Picture 控件显示图像

在 Visual C++中可以使用 Picture 控件静态显示一张图片，即图片先通过资源管理器加载，在程序运行时就已经存在于程序之中，被直接显示出来。其方法如下所述。

技巧：使用 Picture 控件时，必须将图片资源先加载到工程中才能使用。

（1）创建一个基于对话框模式的应用程序——PicDemo，如图 5.1 所示。

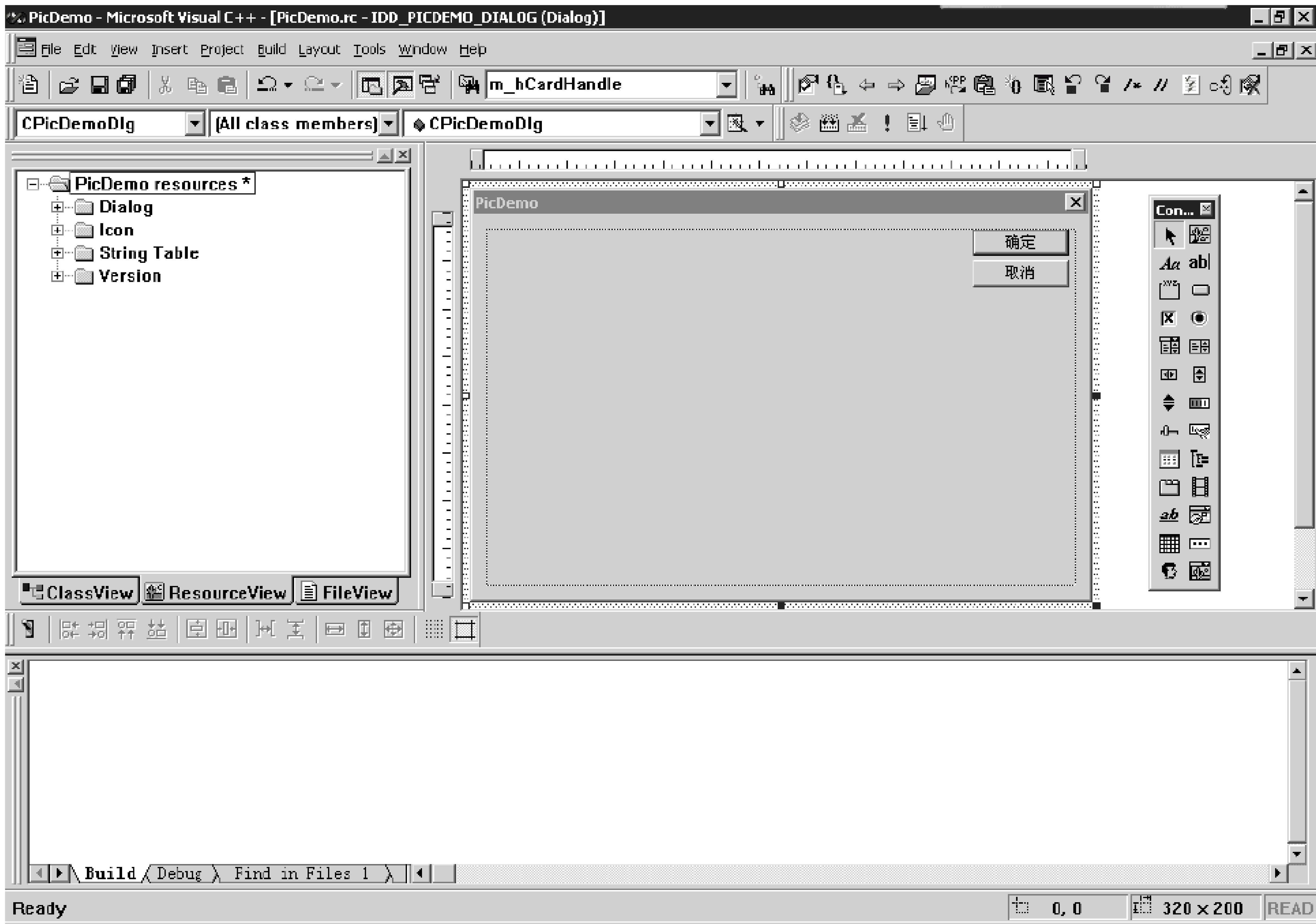


图 5.1　PicDemo 基于对话框模式的主界面

（2）给当前应用程序添加一个图片资源。在图 5.1 的左边树型菜单中，右击 PicDemo resourecs 结点。在弹出的快捷菜单中选中 Import 选项，如图 5.2 所示。

（3）之后会弹出 Import Resource 对话框，如图 5.3 所示。在这里选中示例程序中的 demo.bmp 文件。这张图片资源的 ID 为 IDB_BITMAP1。



图 5.2　选择 Import 选项　　　　图 5.3　Import Resource 对话框

（4）添加图片资源后，需要给对话框添加 Picture 控件。添加方法为：选中右边控件列表中的 Picture 控件，并拖入对话框资源中即可。加入控件后的效果如图 5.4 所示。



图 5.4　添加 Picture 控件

（5）右击对话框上的 Picture 控件。在弹出的菜单中选中 Properties（属性）选项。在弹出的对话框的 Type 下拉列表框中选择 Bitmap，紧跟着下面会出现一个 Image 下拉列表框，如图 5.5 所示。在该下拉列表框中就会看到所有已经载入好的图片，选中 IDB_BITMAP1 的图片资源。

（6）编译程序并运行，最后效果如图 5.6 所示。



图 5.5　Picture 控件的属性对话框设置　　　　图 5.6　静态显示图片程序运行效果

5.3.2　通过对话框背景显示图像

其实在 Visual C++中，还可以直接通过重载对话框的消息函数 OnCtlColor()来实现背景显示图片。方法如下所述。

（1）创建一个基于对话框模式的应用程序——BackPicDemo，如图 5.7 所示。

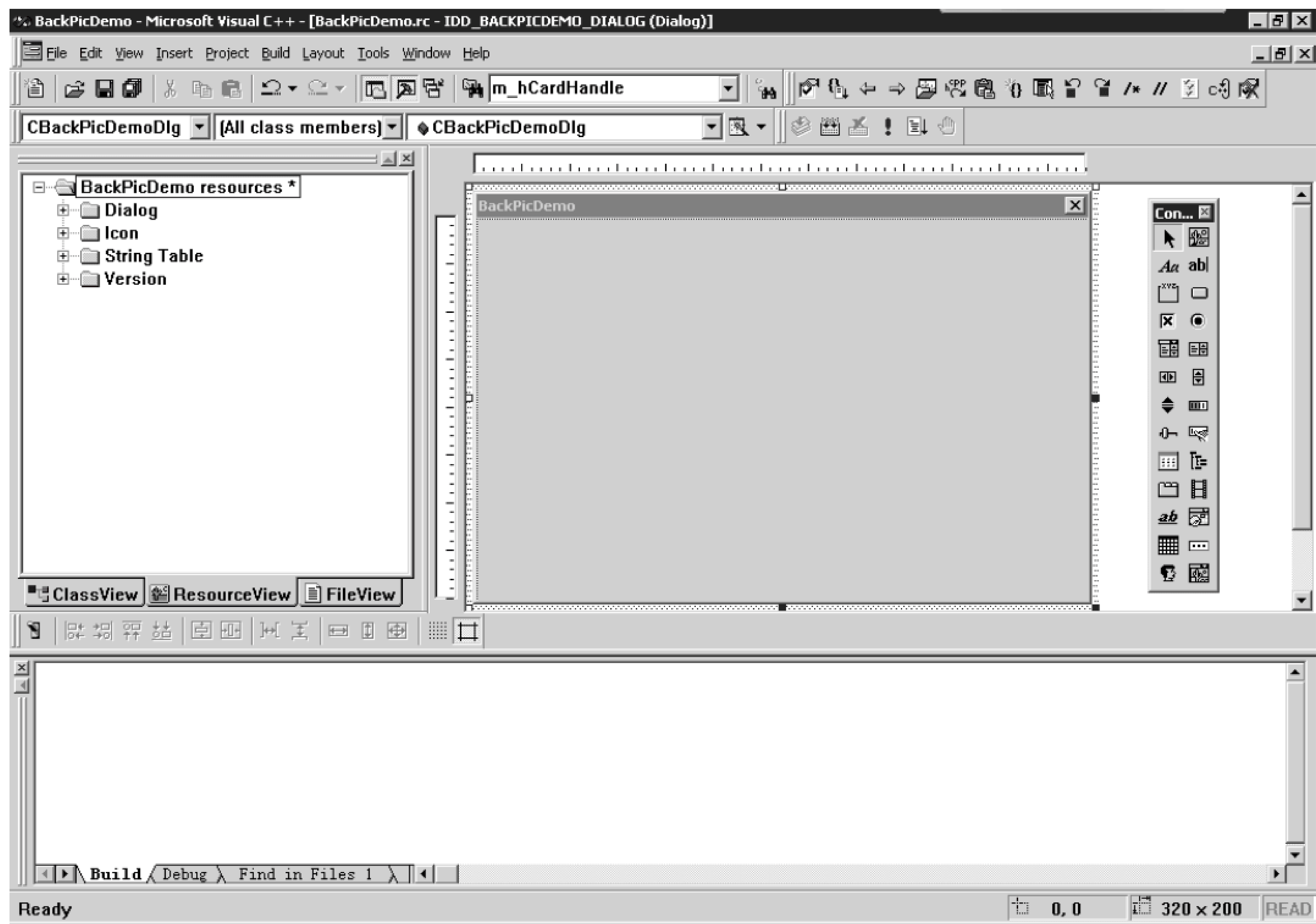


图 5.7　BackPicDemo 程序主界面

（2）给当前应用程序添加一个图片资源。ID 为 IDB_BITMAP1（同前面的示例类似）。

（3）在 CBackPicDemoDlg 类的声明中，给该类增加一个成员变量 m_bkPic，如代码 5.1 所示。

代码 5.1　CBackPicDemoDlg 类的声明

```
01  #if _MSC_VER > 1000
02  #pragma once
03  #endif
04
05  ///////////////////////////////////////////////////////////////////////
06  //CBackPicDemoDlg 对话框代码开始
07
08  class CBackPicDemoDlg : public CDialog           //类声明
09  {
10                                                   //各种声明及定义
11  public:
12      CBackPicDemoDlg(CWnd* pParent = NULL);       //构造函数
13      CBrush m_brBk;                               //笔刷类变量
14                                                   //对话框数据
15      //{{AFX_DATA(CBackPicDemoDlg)
16      enum { IDD = IDD_BACKPICDEMO_DIALOG };       //对话框资源
17      //}}AFX_DATA
```

```
18
19                                                     //虚函数声明
20      //{{AFX_VIRTUAL(CBackPicDemoDlg)
21      protected:
22      virtual void DoDataExchange(CDataExchange* pDX);//资源加载
23      //}}AFX_VIRTUAL
24
25                                                     //各种类的成员
26  protected:
27      HICON m_hIcon;                                 //图标变量
28
29                                                     //消息映射表
30      //{{AFX_MSG(CBackPicDemoDlg)
31      virtual BOOL OnInitDialog();                   //初始化函数
32      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
33      afx_msg void OnPaint();                        //绘图函数
34      afx_msg HCURSOR OnQueryDragIcon();
                                                       //重载的显示函数
35      afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
36      //}}AFX_MSG
37      DECLARE_MESSAGE_MAP()
38  };
39
40  //{{AFX_INSERT_LOCATION}}
41  #endif
```

（4）在 BackPicDemo 类的实现源文件的初始化函数 OnInitDialog()中加入代码，如代码 5.2 所示。

代码 5.2　OnInitDialog()函数实现

```
01  ///////////////////////////////////////////////////////////////////
02  //CBackPicDemoDlg 消息函数的实现
03
04  BOOL CBackPicDemoDlg::OnInitDialog()
05  {                                               //初始化函数实现
06      CDialog::OnInitDialog();
07
08                                                  //比较关于对话框在资源号
09      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
10      ASSERT(IDM_ABOUTBOX < 0xF000);
11                                                  //加载系统菜单
12      CMenu* pSysMenu = GetSystemMenu(FALSE);
13      if (pSysMenu != NULL)
14      {
15          CString strAboutMenu;
16          strAboutMenu.LoadString(IDS_ABOUTBOX);
17          if (!strAboutMenu.IsEmpty())        //关于菜单是否为空
18          {
19              pSysMenu->AppendMenu(MF_SEPARATOR);
20              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                strAboutMenu);
21          }
22      }
23
24      SetIcon(m_hIcon, TRUE);                     //设置大图标
25      SetIcon(m_hIcon, FALSE);                    //设置小图标
26
```

```
27        CBitmap bmp;                                  //图片类对象
28        bmp.LoadBitmap(IDB_BITMAP1);                  //对象加载图片
29        m_brBk.CreatePatternBrush(&bmp);              //创建图片笔刷
30        bmp.DeleteObject();                           //删除对象
31        return TRUE;
32    }
```

代码解析：代码第 27 行是定义图片类对象 bmp。第 28 行通过调用对象的 LoadBitmap()函数加载指定图片资源。代码第 29 行创建包含 bmp 对象图片笔刷。

（5）再打开类向导（可使用快捷键 Ctrl+W），找到 WM_CTLCOLOR 消息，然后单击 Add Function 按钮，如图 5.8 所示。

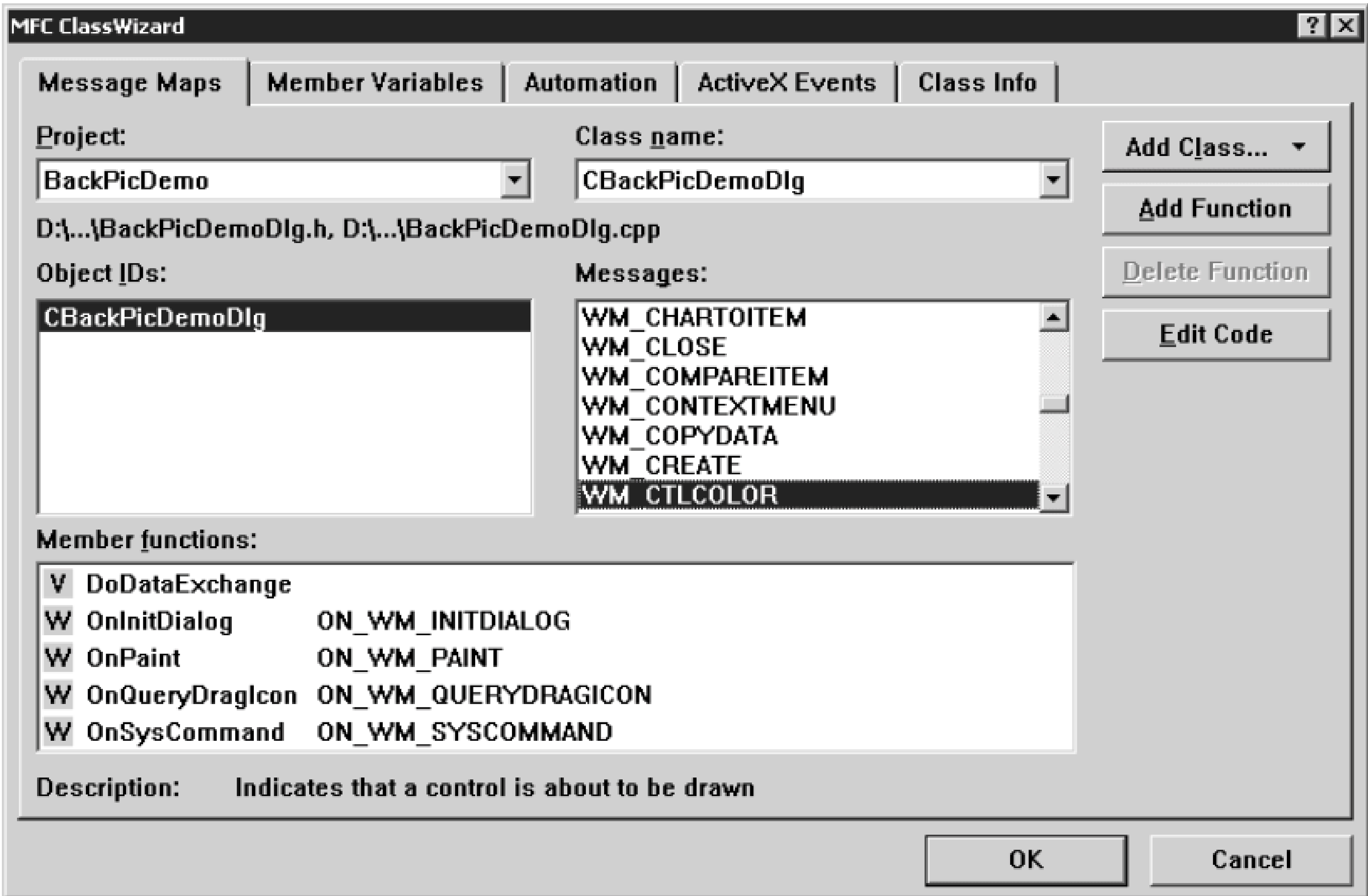


图 5.8　类向导对话框

（6）在 BackPicDemo.cpp 中重载得到对应函数 OnCtlColor()，其实现如代码 5.3 所示。

代码 5.3　OnCtlColor ()函数的实现

```
01  HBRUSH CBackPicDemoDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT
    nCtlColor)
02  {
03      HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
04
05      if (pWnd == this)                   //判断是否是本窗口
06      {
07          return m_brBk;                  //返回笔刷对象的句柄
08      }
09
10      return hbr;                         //返回默认句柄
11  }
```

代码解析：代码第 7 行，将前面创建的图片笔刷返回给调用窗口函数，然后就会将相应的图片刷新到对话框背景上。

（7）程序编译并运行，效果如图 5.9 所示。

从程序最后的效果图中可以看出，使用背景方式显示图片时，当图片面积比对话框面积小时，其会自动把图片平铺到背景上。



图 5.9　BackPicDemo 运行效果

5.3.3　使用 BitBlt()函数动态显示图像

除了前面的两种方法外，还可以使用 BitBlt()函数来实现图像的显示。使用 BitBlt()函数显示图像，是一种动态加载图片的方法，即在程序运行时，指定图片的路径，实现图片的显示。不用在程序生成时，把图片当作资源加载进程序。其方法如下所示。

技巧： 使用 BitBlt()函数时，可以不用把图片作为资源加载到对话框中，而且效率更高。

（1）创建一个基于对话框模式的应用程序——DynamicPicDemo，并在对话框中加入一个 Picture 控件，设置名称为 IDC_PIC，如图 5.10 所示。

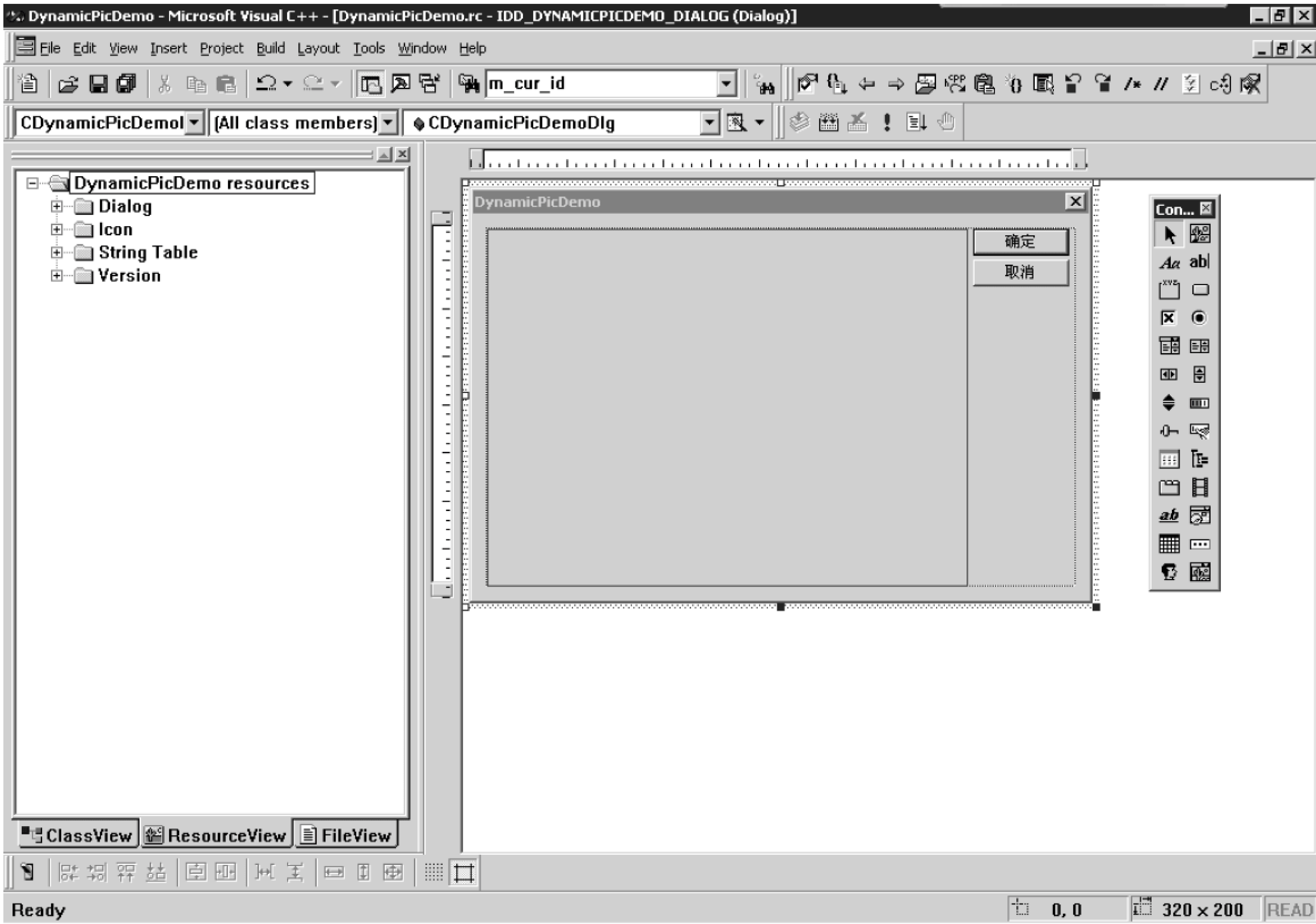


图 5.10　DynamicPicDemo 主界面

（2）在 CDynamicPicDemoDlg 类的声明中，声明一个 CBitmap 类的对象 m_bmp，如代码 5.4 所示。

代码 5.4　CDynamicPicDemoDlg 类的声明

```
01  ///////////////////////////////////////////////////////////////////////////
02  //CDynamicPicDemoDlg 对话框类的声明
03
04  class CDynamicPicDemoDlg : public CDialog
05  {
06                                                  //各种声明及定义
07  public:
08      CDynamicPicDemoDlg(CWnd* pParent = NULL);   //构造函数
09
10      //{{AFX_DATA(CDynamicPicDemoDlg)
11      enum { IDD = IDD_DYNAMICPICDEMO_DIALOG };
12                                                  //资源加载
13      //}}AFX_DATA
14
15      //{{AFX_VIRTUAL(CDynamicPicDemoDlg)         //虚函数声明
16      protected:
17      virtual void DoDataExchange(CDataExchange* pDX);
18      //}}AFX_VIRTUAL
19  private:
20      CBitmap   m_bmp;                    //声明 CBitmap 类的对象 m_bmp
21  protected:
22      HICON m_hIcon;                              //声明图标对象
23      //{{AFX_MSG(CDynamicPicDemoDlg)
24      virtual BOOL OnInitDialog();                //声明初始化函数
25      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
26      afx_msg void OnPaint();                     //声明显示函数
27      afx_msg HCURSOR OnQueryDragIcon();
28      //}}AFX_MSG
29      DECLARE_MESSAGE_MAP()
30  };
```

（3）在对话框的初始化函数（OnInitDialog）中，实现图片文件的加载。添加内容如代码 5.5 所示。

代码 5.5　OnInitDialog()函数的实现

```
01  BOOL CDynamicPicDemoDlg::OnInitDialog()
02  {
03      CDialog::OnInitDialog();                    //初始化对话框函数
04
05      CMenu* pSysMenu = GetSystemMenu(FALSE);
06      if (pSysMenu != NULL)                       //判断是否为空
07      {
08          CString strAboutMenu;
09          strAboutMenu.LoadString(IDS_ABOUTBOX);
10          if (!strAboutMenu.IsEmpty())
11          {                                       //不为空
12              pSysMenu->AppendMenu(MF_SEPARATOR);
13              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                strAboutMenu);
14          }
15      }
16
```

```
17      SetIcon(m_hIcon, TRUE);                         //设置大图标
18      SetIcon(m_hIcon, FALSE);                        //设置小图标
19
20      if( m_bmp.m_hObject != NULL )                   //判断是否已经有对象
21      {
22          m_bmp.DeleteObject();
23      }
24                                                      //载入图片文件
25      HBITMAP hbmp = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
                          ".\\demo.bmp",
                          IMAGE_BITMAP,
                          0,0,
                          LR_CREATEDIBSECTION|LR_LOADFROMFILE);
26  if( hbmp == NULL )
27  {                                                   //判断是否加载成功
28      return FALSE;
29      }
30      //下面的程序用来取得加载的 BMP 图片的信息
31      m_bmp.Attach( hbmp );
32
33      DIBSECTION ds;
34
35      BITMAPINFOHEADER &bminfo = ds.dsBmih;
36
37      m_bmp.GetObject( sizeof(ds), &ds );
38
39      int cx=bminfo.biWidth;                          //得到图像宽度
40
41      int cy=bminfo.biHeight;                         //得到图像高度
42
43      //得到图像的宽度和高度，对图像进行适应，调整控件的大小，让其正好显示一张图片
44
45      CRect rect;
46
47      GetDlgItem(IDC_PIC)->GetWindowRect(&rect);  //得到图像控件的区域
48
49      ScreenToClient(&rect);                          //转换坐标
50                                                      //调整大小
51      GetDlgItem(IDC_PIC)->MoveWindow(rect.left,rect.top,cx,cy,true);
52
53      return TRUE;
54  }
```

代码解析：第 25 行，是创建图片文件句柄，并指向相应的图片文件。第 31 行，用 m_bmp 图片对象加载 BMP 图片的信息，用于第 39 行和第 41 行得到图片的宽度和高度。第 51 行调整图像控件的大小。

（4）上面实现了图片加载，并调整对话框中的图像控件大小以适应图像。现在就差绘制图像的过程。在这里需要打开类向导，重载 WM_PAINT 消息。重载得到对应函数 OnPaint()，修改其实现如代码 5.6 所示。

代码 5.6　OnPaint()函数的实现

```
01  void CDynamicPicDemoDlg::OnPaint()
02  {
03      if (IsIconic())
04      {
05          CPaintDC dc(this);                              //得到设置 DC
```

```
06
07          SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
08
09          int cxIcon = GetSystemMetrics(SM_CXICON);  //得到图标的 x 坐标
10          int cyIcon = GetSystemMetrics(SM_CYICON);  //得到图标的 y 坐标
11          CRect rect;                                //定义矩形区域对象
12          GetClientRect(&rect);                      //得到客户区域
13          int x = (rect.Width() - cxIcon + 1) / 2;
14          int y = (rect.Height() - cyIcon + 1) / 2;
15          dc.DrawIcon(x, y, m_hIcon);                //画图标
16      }
17      else
18      {   //得到 Picture 控件的 DC，图像将被绘制在控件上
19          CPaintDC dc(GetDlgItem(IDC_PIC));
20
21          CRect rcclient;                         //得到客户区域
22          //通过控件句柄得到控件区域
23          GetDlgItem(IDC_PIC)->GetClientRect(&rcclient);
24
25          CDC memdc;                              //内存 DC
26
27          memdc.CreateCompatibleDC(&dc);          //从控件 DC 生成内存 DC
28          CBitmap bitmap;                         //定义图像对象
29          /从内存 DC 创建 bitmap
30          bitmap.CreateCompatibleBitmap(&dc, rcclient.Width(),
            rcclient.Height());
31
32          memdc.SelectObject( &bitmap );          //设置对象
33          //调用默认 OnPaint
34          CWnd::DefWindowProc(WM_PAINT, (WPARAM)memdc.m_hDC , 0);
35          CDC maskdc;
36          maskdc.CreateCompatibleDC(&dc);
37          CBitmap maskbitmap;                     //定义图像对象
38                                                  //创建 bitmap
39          maskbitmap.CreateBitmap(rcclient.Width(), rcclient.Height(),
            1, 1, NULL);
40
41          maskdc.SelectObject( &maskbitmap );     //设置对象
42          maskdc.BitBlt( 0, 0, rcclient.Width(), rcclient.Height(),
            &memdc,                                 //绘制图像到控件区域
43              rcclient.left, rcclient.top, SRCCOPY);
44
45          CBrush brush;                           //定义笔刷变量
46
47          brush.CreatePatternBrush(&m_bmp);       //创建笔刷
48          dc.FillRect(rcclient, &brush);          //填充客户区域
49          //开始绘制真实的图像
50          dc.BitBlt(rcclient.left, rcclient.top, rcclient.Width(),
            rcclient.Height(),
                &memdc, rcclient.left, rcclient.top,SRCPAINT);
51          brush.DeleteObject();                   //删除笔刷对象
52      }
53  }
```

代码解析：第 30 行，从内存 DC 中创建 bitmap 对象。第 39 行，用于创建 bitmap。第 42 行，调用 BitBlt()函数绘制图像到控件区域。

（5）程序编译并运行，其效果如图 5.11 所示。



图 5.11　DynamicPicDemo 运行效果

从最后的执行效果上看，和图 5.6 相似。不过读者在这里需要注意，使用 BitBlt()函数时，图片资源是从文件中得到的，而不是作为程序资源，在生成程序时已经存在。这样做的好处是使程序变得小巧、灵活，图片也可以随时替换。

技巧： *BitBlt()函数可以在对话框的任何位置画图，但要记住显示屏的 X 坐标是从左到右增加，Y 坐标是从上向下增加。*

5.4　游戏中音乐的播放

在玩游戏时，除了漂亮丰富的图像能够吸引人外，优美动听的声音同样也可以使游戏更加精彩。那么在游戏中，声音是如何加载进来？又如何播放出来的呢？在本节中，将向大家简单介绍。

在游戏中要播放声音，可以调用 Windows 的 API 函数 sndPlaySound 来实现。这个函数的原型如下：

```
BOOL WINAPI sndPlaySound(LPCSTR pszSound, UINT fuSound);
```

其中，pszSound 是指定将要播放声音文件的路径；fuSound 指定播放声音的方式，其可选参数如表 5.1 所示。

表 5.1　sndPlaySound中参数fuSound的参数值表

宏　定　义	说　　明
SND_ASYNC	在播放的同时继续执行后面的语句
SND_LOOP	一直重复播放声音，直到下一次调用本函数
SND_MEMORY	把声音数据载入内存
SND_NODEFAULT	指如果没有找到声音文件，不播放默认声音
SND_NOSTOP	不停止当前播放的声音
SND_SYNC	播放完声音之后再执行后面的语句

🔔**技巧：**使用 API 函数时，要注意添加相应的头文件及静态库文档，否则在编译或者连接时会产生错误。

下面就跟随笔者来创建一个有声音的应用程序——PlaySoundDemo，其功能要求为：单击“播放声音”按钮时，可以播放一段声音，而单击“播放音乐”按钮时，可以播放一段音乐。创建应用程序步骤如下所述。

（1）创建基于对话框的应用程序——PlaySoundDemo，并添加“播放声音”和“播放音乐”两个按钮。ID 号分别为 IDC_PLAY_SOUND 和 IDC_PLAY_MUSIC。主界面如图 5.12 所示。

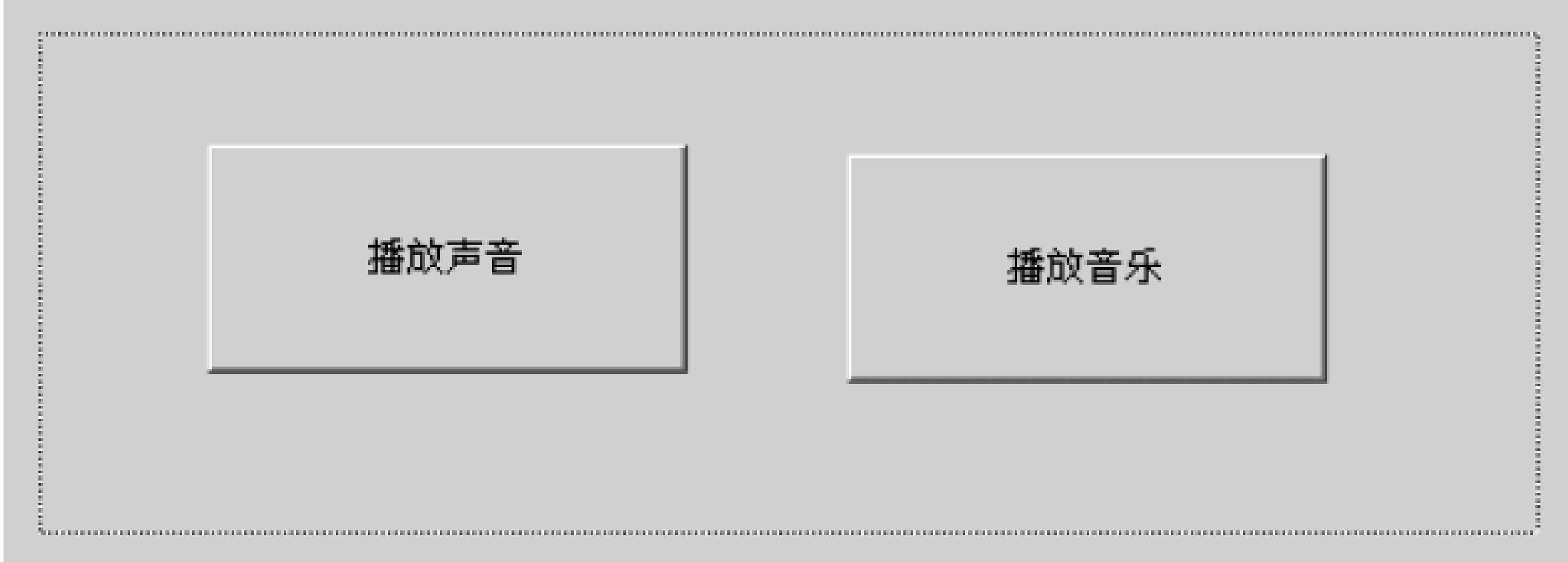


图 5.12　PlaySoundDemo 主界面

（2）用类向导（可以用快捷键 Ctrl+W 启动）给 IDC_PLAY_SOUND 和 IDC_PLAY_MUSIC 两个按钮资源添加响应函数，添加方法为：找到 IDC_PLAY_SOUND 选项，并单击 Add Function 按钮，如图 5.13 所示。

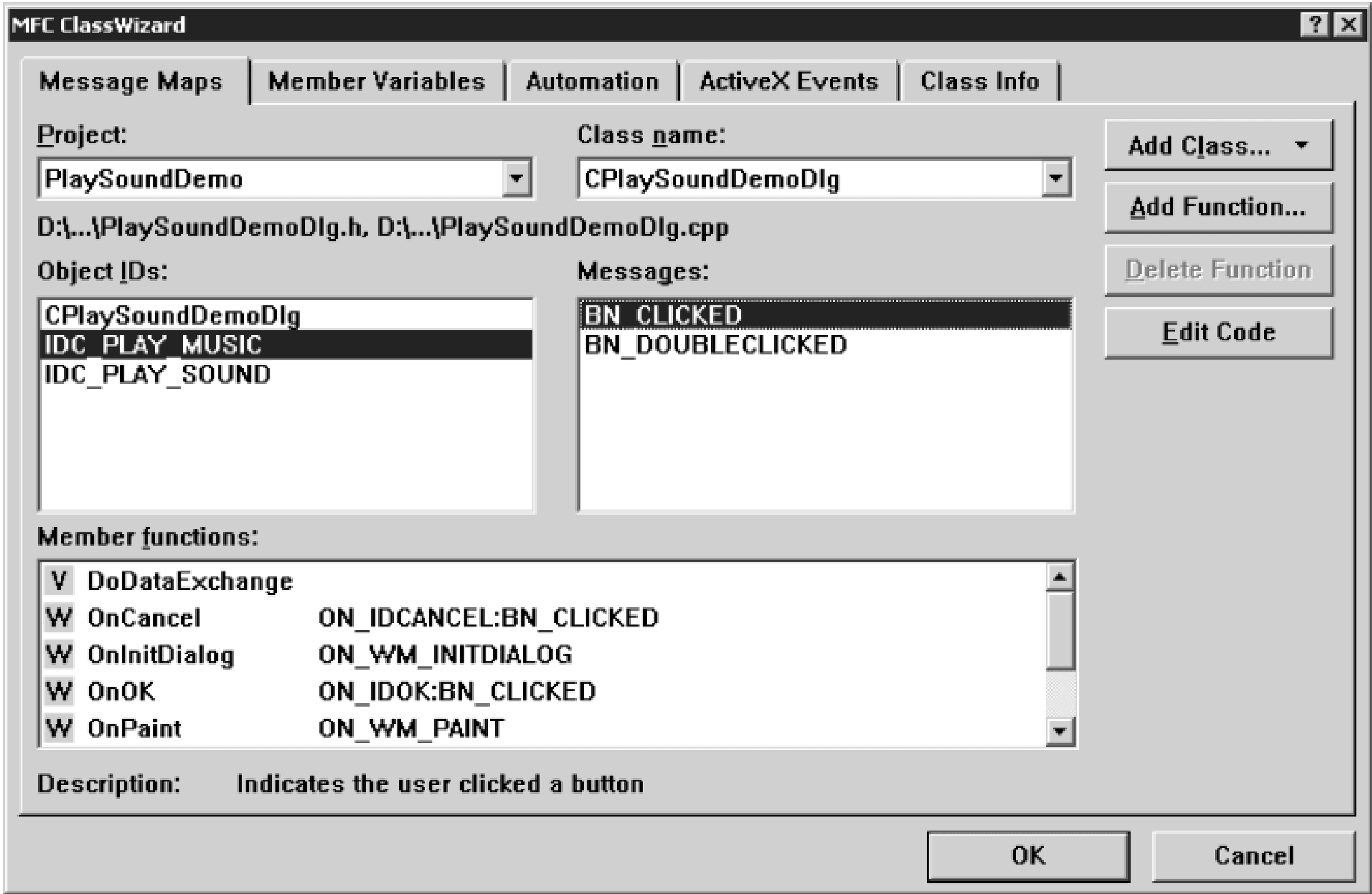


图 5.13　添加响应函数的类向导

（3）添加完成后，分别在这两个按钮的实现函数中增加代码，如代码 5.7 所示。

代码 5.7　按钮响应函数的实现

```
01  void CPlaySoundDemoDlg::OnPlaySound()
02  {    //指定文件并播放
03       sndPlaySound("ding.wav",SND_ASYNC);
04  }
05
06  void CPlaySoundDemoDlg::OnPlayMusic()
07  {    //指定文件并播放
08       sndPlaySound("music.wav", SND_ASYNC);
09  }
```

代码解析：第 3 行和第 8 行调用 sndPlaySound()函数播放指定文件。

（4）在使用该 API 函数时，还需要在文件的开始加入函数的声明，其函数声明在 mmsystem.h 文件中。插入头文件的方法如下：

```
#include <mmsystem.h>                    //插入头文件
```

（5）除了增加头文件外，还需要在当前工程中加入 winmm.lib 静态库文件。添加方法为选择 Project | Settings 命令，如图 5.14 所示。

（6）在弹出的对话框中，选择 Link 标签，进入 Link 选项卡。在 object/library modules 文本框中输入 winmm.lib 文件名，如图 5.15 所示。

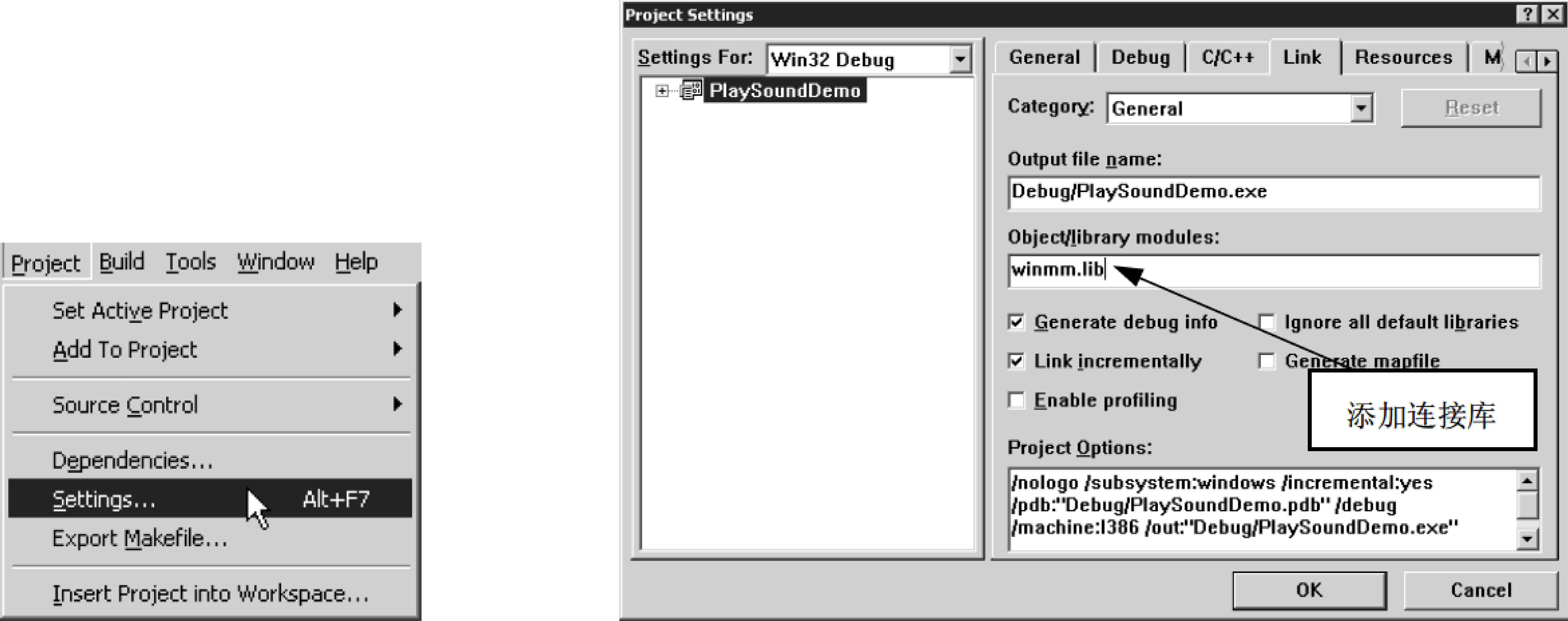


图 5.14　选中 Settings 菜单项　　　　图 5.15　添加静态库文件

（7）编译并执行程序，结果如图 5.16 所示。要注意，因为这是播放声音和音乐，所以只能用听来测试效果。



图 5.16　程序运行效果图

5.5　游戏中的互动

前面讲解的内容都是由电脑输出的，用户接收的是单向的。要实现人机交互，必须是用户也能够把数据输入到电脑中。所以本节将简单介绍如何实现电脑游戏的输入处理。

5.5.1　系统对输入设备的处理

当读者在键盘上按下一个键时，电脑屏幕上就会出现一个对应的字符。这种交互方式

看上去非常直观和简单，但其实键盘跟系统之间的交互却是非常烦琐的。作为游戏程序员，必须理解系统对输入设备的处理方法，才能为以后的游戏开发扫平障碍。

每当按下或者释放一个键时，一个数字信号就会被传送给键盘的微处理器。随后这个键盘微处理器就会向电脑系统“申请”一个中断，同时系统从键盘那里获得了一个字符码，从而使得系统得知到底是哪个键被按下或释放。微处理器给电脑系统传送的那个字符码被称作扫描码。

这里需要指出的是一个扫描码的大小是一个字节（8 位），其中低 7 位（即 bit0-6）表示哪个键被操作，而最高位代表是键被按下还是被释放。所以电脑中所能处理的最多的键的数目是 128 个。

那么 Windows 中的键盘处理又是如何的呢？其实是通过 Windows 的消息机制来实现对键盘输入的处理。其过程如下所述。

（1）Windows 系统把扫描码转换为虚拟码和 ASCII 码。虚拟码只是将原来的扫描码在 Windows 里进行了包装，用 VK_A 而不是 ASCII 码的“30”来表示 A。而 ASCII 码是为了实现扫描码和字符之间的对应关系。在 ASCII 码中 A（0x41）和 a（0x61）所对应的 ASCII 码是不同的。但最多也只能表示 128 种不同的字符。

在 Windows 中为了能表示更多的字符，有时要用到扩展的 ASCII 码。所谓扩展就是增添了一位附加信息，这样就使得可以表示的字符数目达到了 256 个。但是仍然不能达到要求，这里就需要用 Unicode 码，在 Unicode 中每个字符用 16 个比特位（2 个字节）来表示，所以总共能表示 65535 种字符，这样就满足了目前的所有需求。

（2）通过消息机制来告诉程序员某个键被按下或者释放了。

（3）对于接收到的虚拟码或者 ASCII 码如何处理就取决于程序员。如果是想用来做文字处理，那么就只需要把字符插入到编辑区域中即可；但对于游戏来说，大多时候键盘是用来控制游戏中的各种角色操作的。

除了键盘处理外，Windows 中还包括鼠标输入的处理。鼠标相对于键盘来说就更加简单了。因为鼠标上的“零件”实在是太少。当读者按下一个键时就给系统发送一个信号，释放时同样要向系统发送信号。

鼠标每隔一个很小的时间间隔就向系统自动报告它的移动信息等数据。鼠标的驱动程序读入这些数据然后转换成相应的形式。因为每个鼠标消息都要传送给消息处理过程，然后再被插入到相应的消息队列等待处理，所以用消息机制来处理鼠标消息是很慢的，但对于一般的游戏应用是能够满足的。

大多数的电脑只有两种输入设备：一种是键盘，另一种是鼠标。所以下面将只对这两种输入设备的处理进行讲解。

5.5.2　键盘消息响应

通过前面知识的学习，相信读者已经对 Windows 处理键盘输入的过程有了一些了解。为了加深对消息机制的理解，在这里笔者给出一个 Windows 处理键盘消息的示例应用程序。其应用程序的需求如下：

- 能够接收到在应用程序窗口中输入的键盘的按键数据。
- 当一个按键（只包括 A～Z）被按下时，在主窗口中输出“XXX 按键被按下”字

符串提示信息。

- 当一个按钮被释放时，在主窗口中输出“XXX 按键被释放”字符串提示信息。

为了程序操作和显示的简单，在这里笔者创建了一个 Windows 窗口应用程序——KeyMessageDemo。其创建过程如下所述。

（1）参照 2.5.3 节所示的方法，创建窗口应用程序项目（KeyMessageDemo），并按规定配置好，即采用默认设置，并把不同类型的文件放置在不同的文件夹内。创建过程如图 5.17 所示。

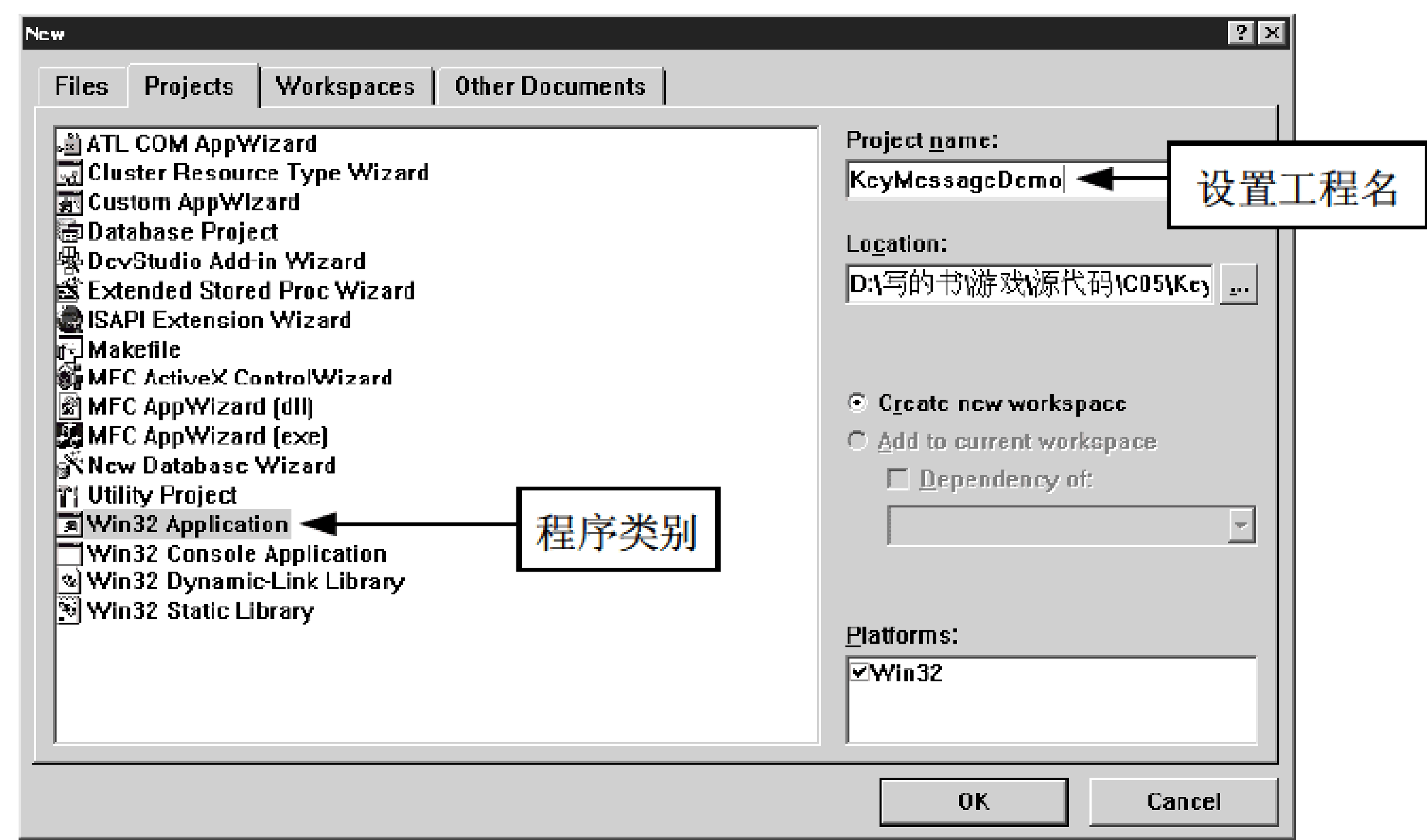


图 5.17　创建 KeyMessageDemo 窗口应用程序项目

（2）在 KeyMessageDemo.cpp 文件中，添加内容如代码 5.8 所示。

代码 5.8　KeyMessageDemo 应用程序的实现

```
//KeyMessageDemo.cpp 文件，包含 KeyMessageDemo 对话框的实现
//
01  #include "stdafx.h"                     //插入头文件
/***************************************************************************/
02  #include <windows.h>                    //一个 Windows 应用程序应该包含的头文件
03  #include <stdio.h>                      //标准输入输出流文件
04  LRESULT CALLBACK WinSunProc
(
HWND hwnd,                                  //窗口句柄
UINT uMsg,                                  //真实消息
WPARAM wParam,                              //消息参数 1
LPARAM lParam                               //消息参数 2
);
 /**************************************************************************
 WinMain:Windows 程序的入口函数
 WINAPI :在应用程序回调函数中作为一个返回值的样式
 当 Windows 的外壳（Windows9X 的资源管理器）侦测到使用者意欲执行一个 Windows 程序，
 于是调用加载器把该程序加载，然后调用 C startup code，后者再调用 WinMain，开始
 执行程序。WinMain 的四个参数由操作系统传递进来
 **************************************************************************/
05  int WINAPI WinMain
06  (
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow
07  {
```

```
    /*************************************************************************/
    /* 创建一个完整的窗口需要经过下面 4 个操作步骤：
    设计一个窗口类；
    注册窗口类；
    创建窗口；
    显示及更新窗口                                                          */
    /*************************************************************************/
08  WNDCLASS wndcls;

09  wndcls.cbClsExtra=0;                  //指定额外内存空间

10  wndcls.cbWndExtra=0;                  //指定额外内存空间
    //指定窗口背景色
11  wndcls.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    //设置光标样式
12  wndcls.hCursor=LoadCursor(NULL,IDC_CROSS);
    //设置图标样式
13  wndcls.hIcon=LoadIcon(NULL,IDI_ERROR);
    //指定窗口实例句柄
14  wndcls.hInstance=hInstance;
    //wndcls.lpfnWndProc 所指定的函数就是窗口的行为中枢也就是所谓的窗口函数
15  wndcls.lpfnWndProc=WinSunProc;
    //窗口类名称，其同时也就是 CreateWindow()函数的第一个参数
16  wndcls.lpszClassName="KeyMessageDemo";
17  wndcls.lpszMenuName=NULL;             //菜单
    /*窗口的显示类型：窗口水平重画*/
18  wndcls.style= CS_HREDRAW|CS_VREDRAW;
    /*注册窗口类*/
19  RegisterClass(&wndcls);
20  HWND hwnd;
    /**************************************************************************
    CreateWindow 只产生窗口，并不显示窗口
    *************************************************************************/
21  hwnd=CreateWindow
        (
        "KeyMessageDemo",                 //已注册窗口类的名称
        "KeyMessageDemo",                 //窗口标题
        WS_OVERLAPPEDWINDOW,              //窗口风格
        200,                              //窗口位置的横坐标
        200,                              //窗口位置的纵坐标
        600,                              //窗口的宽度
        400,                              //窗口的高度
        NULL,
        NULL,
        hInstance,                        //实例句柄
        NULL
        );
    /*显示窗口*/
22  ShowWindow(hwnd,SW_SHOWNORMAL);
23  UpdateWindow(hwnd);                   /*更新窗口*/
    /************************************************************************/
    /* 初始化工作完成后，WinMain 进入所谓的消息循环*/
    /************************************************************************/
24  MSG msg;
25  while(GetMessage(&msg,NULL,0,0))
26  {
```

```
27      TranslateMessage(&msg);          //转换键盘消息
28      DispatchMessage(&msg);           //分派消息
29   }
30   return 0;
31   }
/*****************************************************************************/
/*窗口函数。窗口函数通常利用 switch/case 方式判断消息的种类，以决定处置方式，由于其是
被 Windows 系统所调用的，所以这是一种 call back 函数，意思是指在程序中，被 Windows 系
统调用的函数，这些函数虽然由用户设计，但是永远不会也不该被用户调用，其是为 Windows 系
统准备的*/
/*****************************************************************************/
32   LRESULT CALLBACK WinSunProc(
        HWND hwnd,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam
     )
33   {
34    char szStr[64] = {0};
35    HDC hdc;
36    switch(uMsg)
37    {
38       case WM_KEYUP:               //当键盘释放时产生 WM_KEYUP 消息
39           hdc=GetDC(hwnd);
40           sprintf(szStr, "%c 按键被释放\r", wParam);
41           TextOut(hdc,100,50,szStr,strlen(szStr));
42           ReleaseDC(hwnd,hdc);
43           break;
44       case WM_KEYDOWN:             //当键盘按下时产生 WM_KEYDOWN 消息
45           hdc=GetDC(hwnd);
46           sprintf(szStr, "%c 按键被按下\r", wParam);
47           TextOut(hdc,100,0,szStr,strlen(szStr));
48           ReleaseDC(hwnd,hdc);
49           break;
50       case WM_CLOSE:
51           if(IDYES==MessageBox(hwnd,
52           "是否真的结束?","KeyMessageDemo",MB_YESNO))
53           {
54               DestroyWindow(hwnd);
55           }
56           break;
57       case WM_DESTROY:
58           PostQuitMessage(0);          //发送退出消息到窗口
59           break;
60       default:
61       return DefWindowProc(hwnd,uMsg,wParam,lParam);
62    }
63    return 0;
64   }
```

代码解析：在这段程序中，第 38～43 行代码是对按键释放的响应实现，即输出一个字符串到窗口上。第 44～49 行代码，是对按键按下的响应实现，同按键释放一样。

技巧：在响应函数中，只能得到相应键盘输入的键盘码，不是字符或者字符串。

（3）编译并执行程序，效果如图 5.18 所示。



图 5.18　KeyMessageDemo 窗口应用程序执行效果

5.5.3　鼠标消息响应

知道了如何处理键盘消息后，现在再来学习如何处理另一个输入设备的消息，即鼠标输入的消息响应。在这里，笔者还是先给出一个简单的开发需求，然后再根据需求进行设计和实现。同时为了让各位读者能够学会如何使用 MFC 中框架应用程序的创建，所以在本示例中加入了 MFC 框架应用程序向导的设置介绍。

应用程序的需求如下：

- 使用 MFC 框架结构式的应用程序界面，能够接收并处理鼠标设备的输入消息。
- 能够将对当前鼠标在窗体上的坐标进行追踪并显示。

为了满足这些要求，笔者创建一个基于 MFC 框架结构的应用程序——MouseTrace。其实现过程如下所述。

（1）创建一个基于 MFC 框架结构的应用程序（MouseTrace），设置程序类型为 Single document 单文档类型。在这个界面的左边，每个类型都有对应的应用程序的界面结构图说明，如图 5.19 所示。

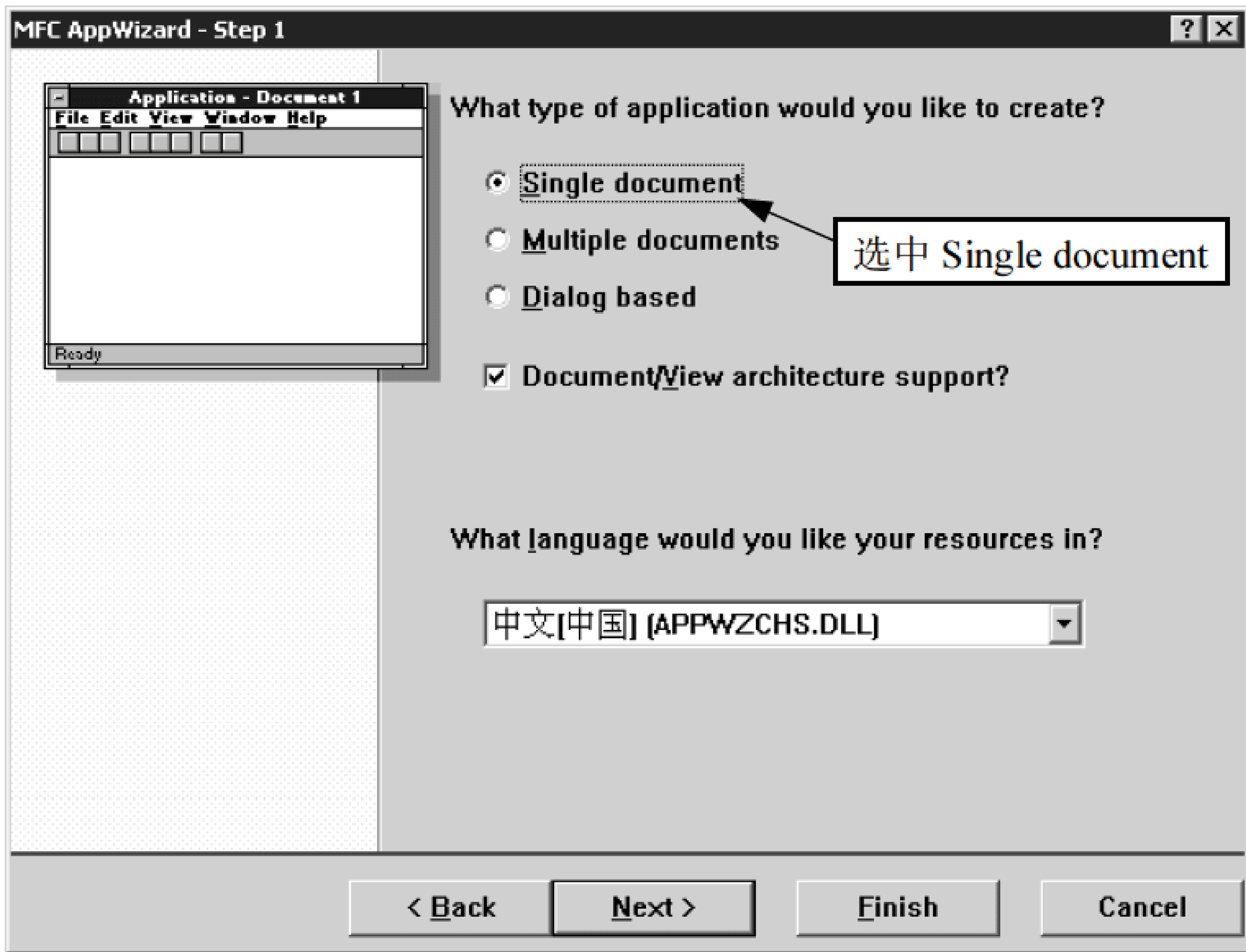


图 5.19　设置程序类型对话框

（2）单击 Next 按钮，进入数据库来源设置对话框。因为这里不需要数据库的支持，所以采用默认设置，如图 5.20 所示。

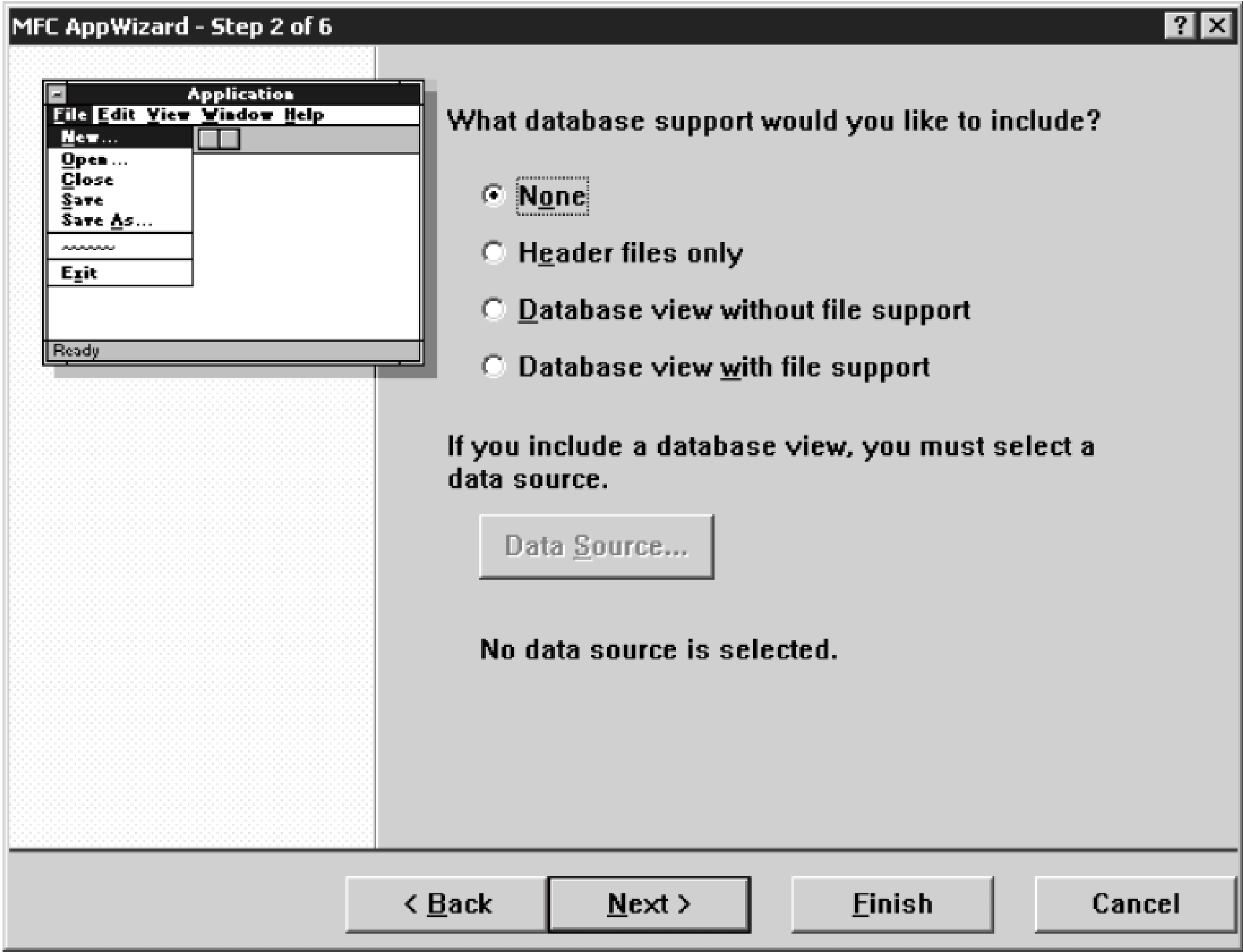


图 5.20　设置数据库来源对话框

对话框中的各选项解释如下。

- None 选项：不引入数据库（默认）。
- Header files only 选项：生成头文件。
- Database view without file support 选项：无文件支持的数据库视图。
- Database view with file support 选项：带文件支持的数据库视图。

对于每种类型，在左边的图示中均有演示。当用户选择数据库支持时，还需要单击 Data Source 按钮设置数据源。

（3）单击 Next 按钮，进入所支持的文档类型设置对话框。本示例中因为不涉及文档，采用默认设置，如图 5.21 所示。对话框中的各选项解释如下所述。

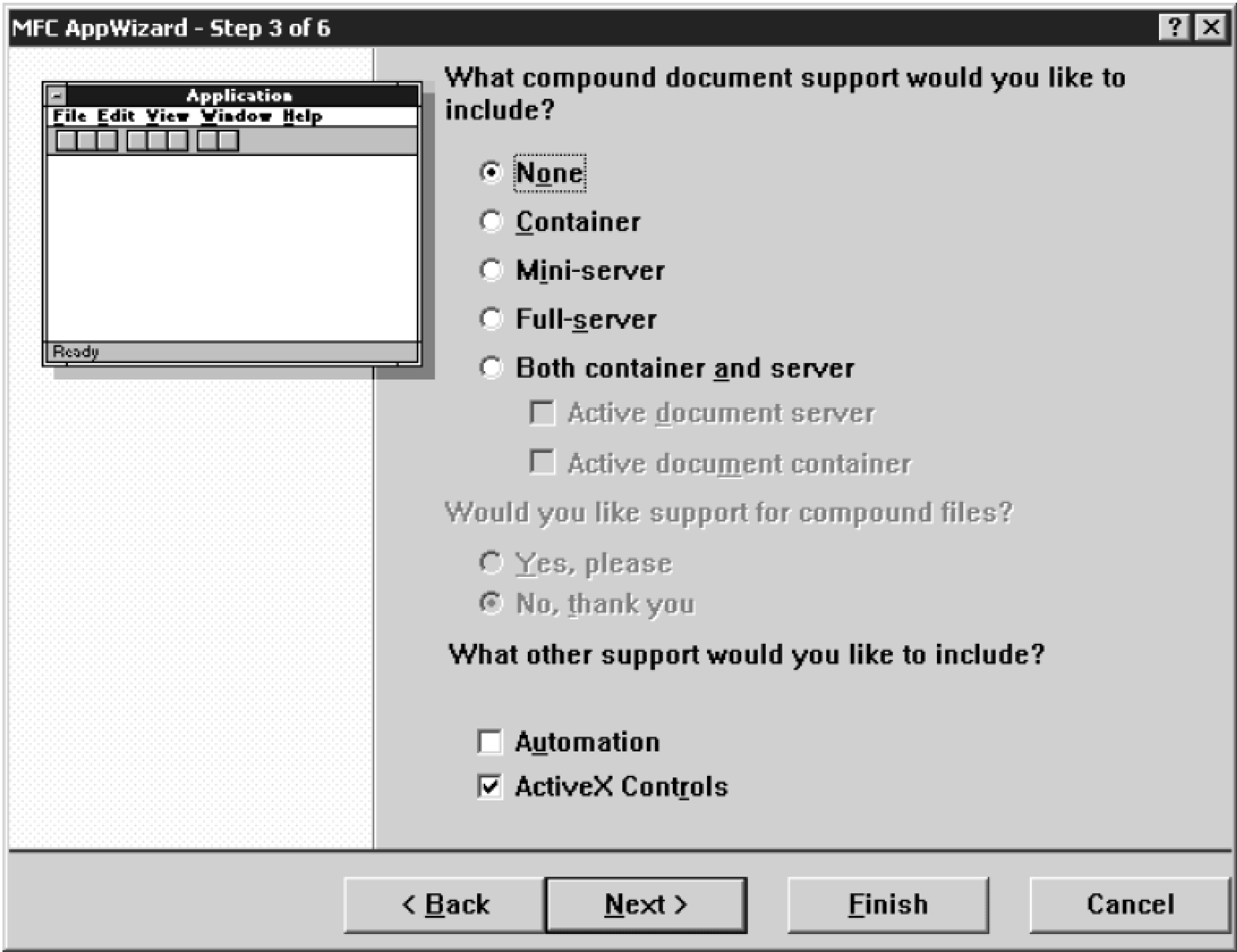


图 5.21　支持的文档类型设置对话框

- None 选项：不支持复合文档（默认）。
- Container 选项：能容纳嵌入的对象或者链接的文档。
- Mini-server 选项：能创建、管理复合文档，支持嵌入。

- ❑ Full-server 选项：能创建、管理复合文档，支持嵌入和链接。
- ❑ Both container and server 选项：容纳文档，并作为服务器管理文档。
- ❑ Automation 选项：自动完成相关操作。
- ❑ ActiveX Controls 选项：支持使用 ActiveX 控件。

（4）单击 Next 按钮，进入程序特征设置对话框。在这个对话框中，用户可以选定应用程序的外观特征，而且也有相应的图示。本示例中，只需要选中 3D controls 和 Initial status bar 两个选项，如图 5.22 所示。

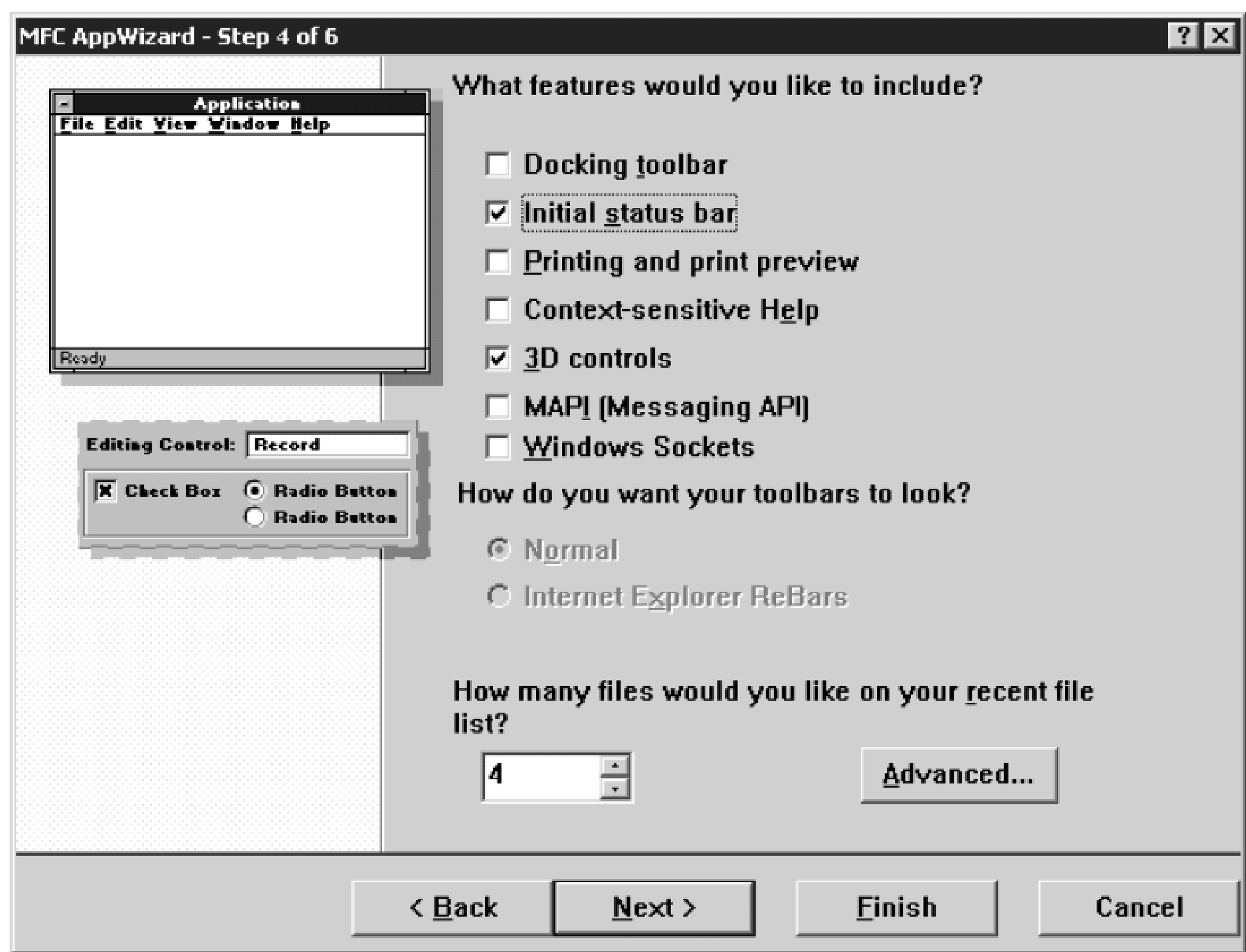


图 5.22　程序特征设置对话框

对话框中的各选项解释如下所述。

- ❑ Docking toolbar 选项：包含工具栏。
- ❑ Initial status bar 选项：包含状态栏。
- ❑ Printing and print preview 选项：包含打印与打印预览。
- ❑ Context-sensitive Help 选项：包含上下文相关的帮助文档。
- ❑ 3D controls 选项：三维效果。
- ❑ MAPI（Messaging API）选项：操作邮件。
- ❑ Windows Sockets 选项：网络 TCP/IP 通信。

（5）单击 Next 按钮，进入应用程序样式设置对话框。本示例中采用默认的设置，如图 5.23 所示。

（6）单击 Next 按钮，进入最终类信息调整对话框。在这里用户可以调整类的相关信息，例如，可以指定每个类的基类、生成的最终文件名、头文件名。本示例中不需要修改这些信息，所以采用默认选项，如图 5.24 所示。

（7）单击 Finish 按钮完成设置。最后会出现信息报告对话框，如图 5.25 所示。单击 OK 按钮后，Visual C++就会立刻创建一个基于框架结构的应用程序。

（8）启动类向导，给 CMouseTraceView 类增加一个消息响应函数 OnMouseMove()，用于响应鼠标在窗口中移动。方法为：在 Class name 下拉列表框中选中 CmouseTraceView 项。同时在 Object IDs 列表框中选中 CMouseTraceView 项。选中 Message 列表框中的 WM_

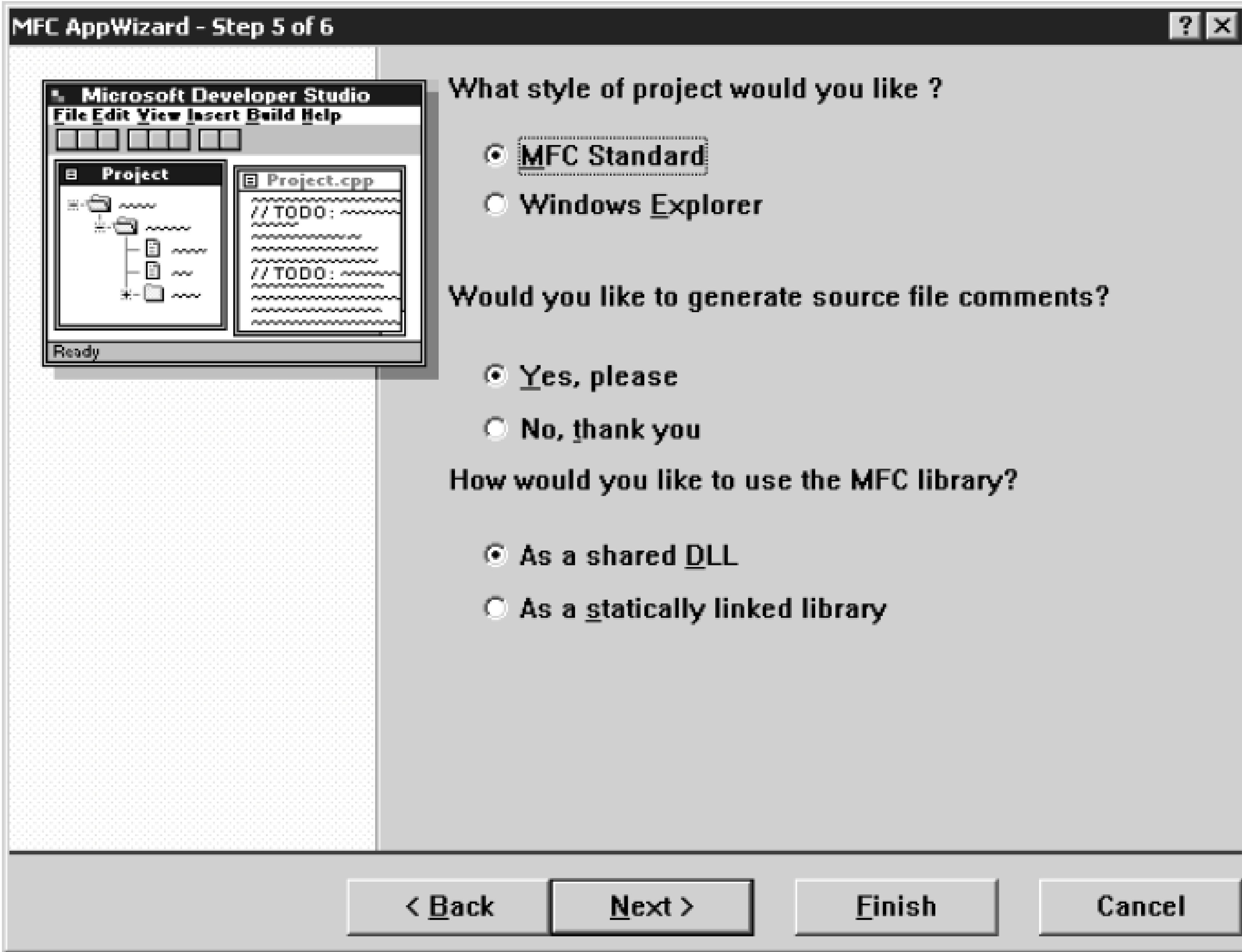


图 5.23　应用程序样式设置对话框

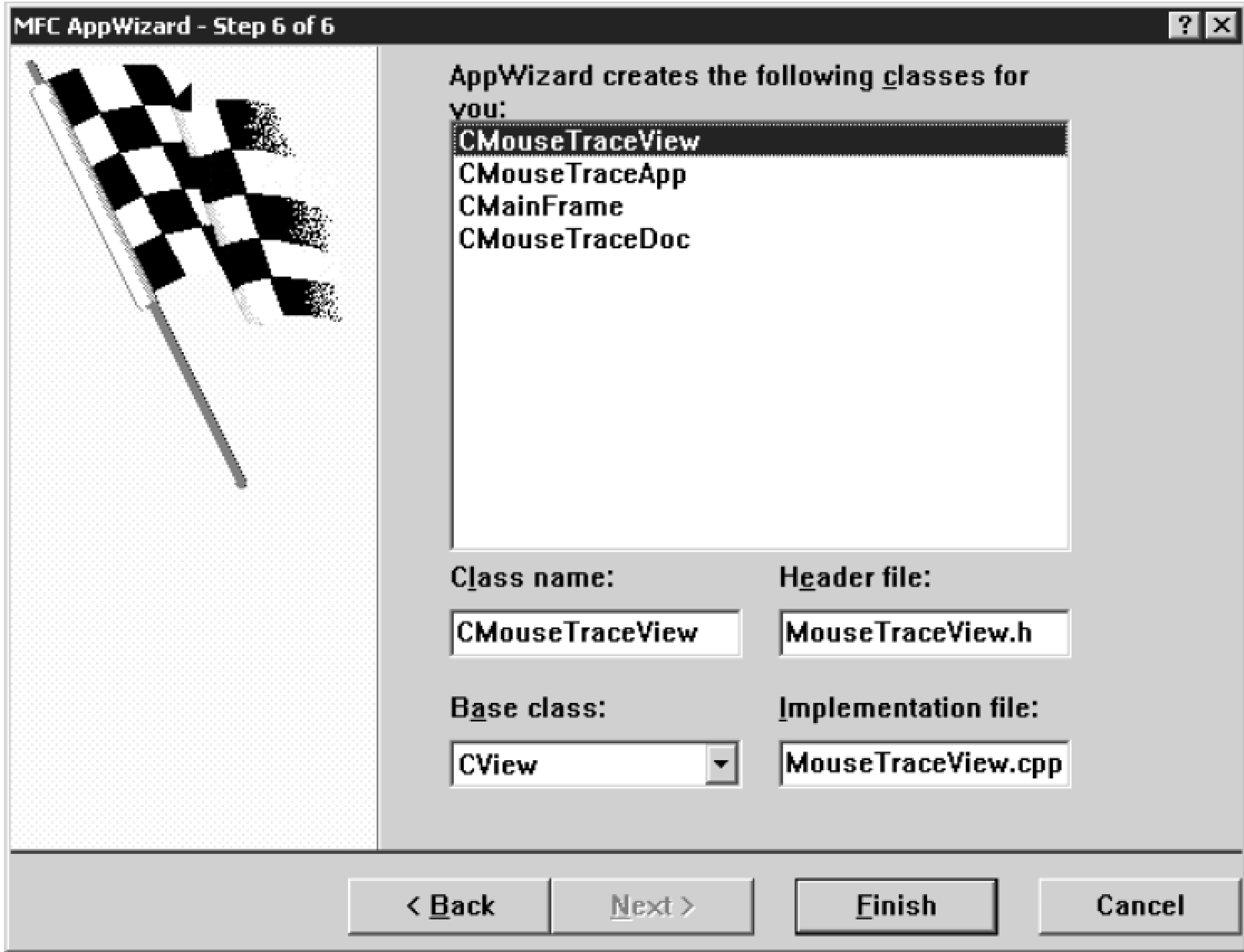


图 5.24　最终类信息调整对话框

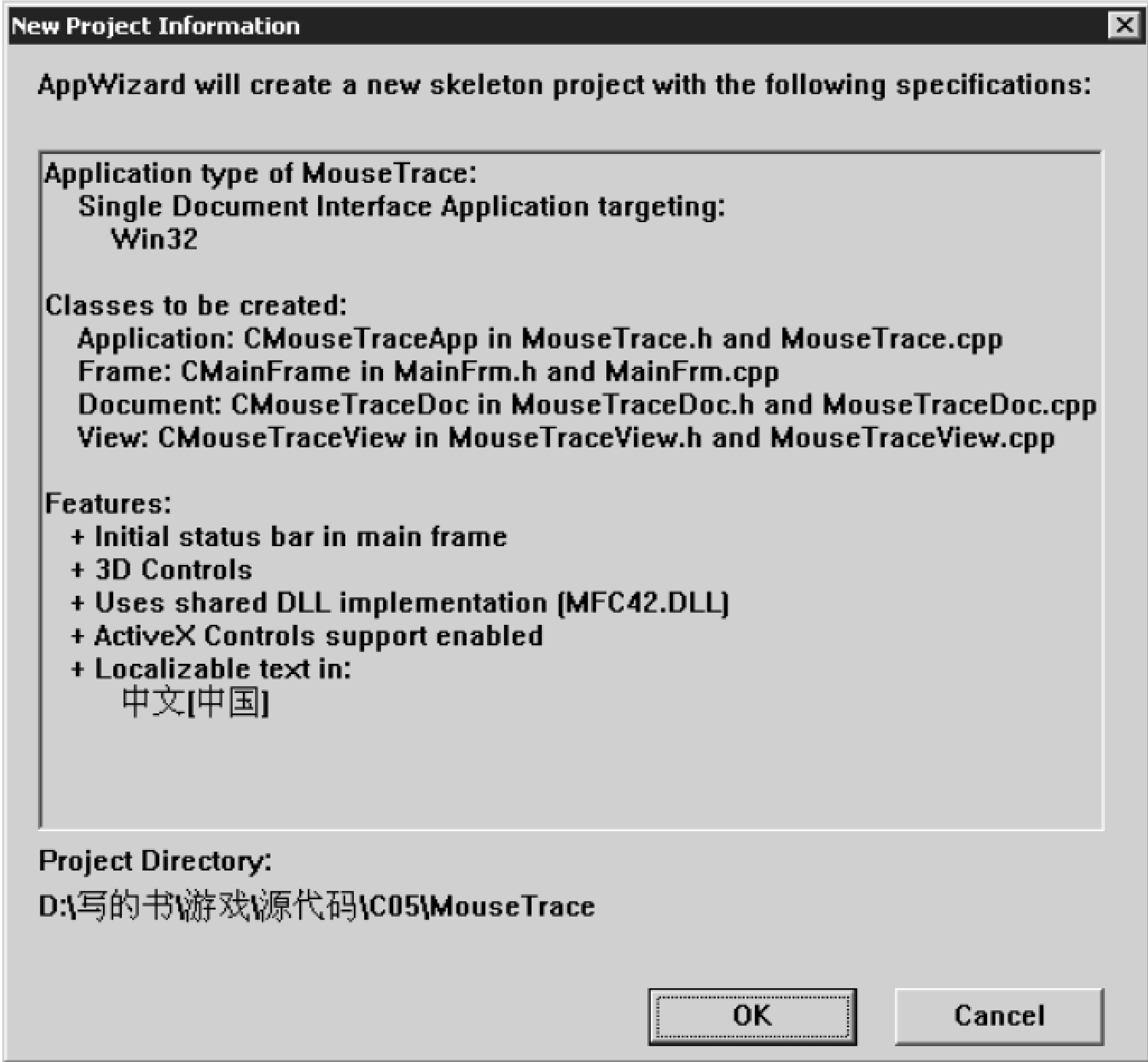


图 5.25　信息报告对话框

MOUSEMOVE 项。然后单击 Add Function 按钮，如图 5.26 所示。

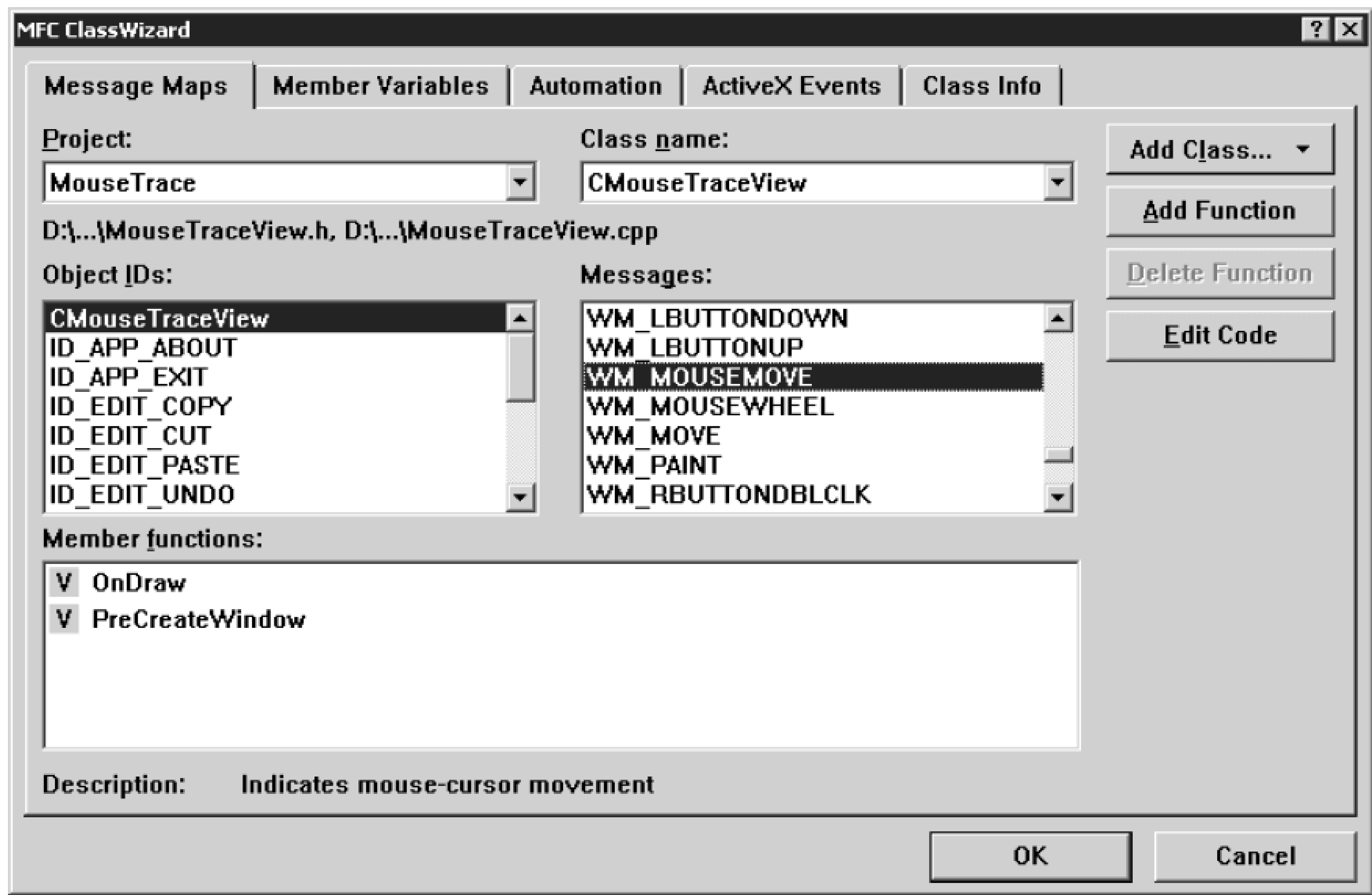


图 5.26　使用类向导添加消息响应函数

（9）在 CMouseTraceView 类声明中，增加一个字符串变量 m_showMsg，用于保存鼠标在窗口的坐标位置，代码如代码 5.9 所示。

代码 5.9　CMouseTraceView 类的声明

```
01  class CMouseTraceView : public CView
02  {
03  protected:
04      CMouseTraceView();                  //构造函数
05      DECLARE_DYNCREATE(CMouseTraceView)
06
07  public:
08      CMouseTraceDoc* GetDocument();      //得到当前文档指针的成员函数声明
09  private:
10      CString m_showMsg;                  //字符串变量，用于保存坐标值
11  public:
12
13      public:
14      virtual void OnDraw(CDC* pDC);      //绘图函数
15      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
16  public:
17      virtual ~CMouseTraceView();         //析构函数
18  #ifdef _DEBUG
19      virtual void AssertValid() const;
20      virtual void Dump(CDumpContext& dc) const;
21  #endif
22
23  protected:
24                                          //鼠标移动消息响应函数声明
25      afx_msg void OnMouseMove(UINT nFlags, CPoint point);
26      DECLARE_MESSAGE_MAP()
27  };
28
29  #ifndef _DEBUG
30  inline CMouseTraceDoc* CMouseTraceView::GetDocument()
31  { return (CMouseTraceDoc*)m_pDocument; }
32  #endif
```

（10）在 CMouseTraceView 类的实现文件中，修改其内容如代码 5.10 所示。

代码 5.10　CMouseTraceView 类的实现

```
01  //MouseTraceView.cpp : CMouseTraceView 类的实现
02
03
04  #include "stdafx.h"                     //插入头文件
05  #include "MouseTrace.h"                 //插入头文件
06
07  #include "MouseTraceDoc.h"
08  #include "MouseTraceView.h"             //插入类声明头文件
09
10  ///////////////////////////////////////////////////////////////////
11  //CMouseTraceView    实现代码开始
12
13  IMPLEMENT_DYNCREATE(CMouseTraceView, CView)
14
15  BEGIN_MESSAGE_MAP(CMouseTraceView, CView)
16      //{{AFX_MSG_MAP(CMouseTraceView)
17      ON_WM_MOUSEMOVE()
18      //}}AFX_MSG_MAP
19  END_MESSAGE_MAP()
20
21  ////////////////////////////////////////////////////////////////////
22  //CMouseTraceView 各成员函数的实现
23
24  CMouseTraceView::CMouseTraceView()
25  {                                   //构造函数的实现
26  }
27
28  CMouseTraceView::~CMouseTraceView()
29  {                                   //析构函数的实现
30  }
31
32  BOOL CMouseTraceView::PreCreateWindow(CREATESTRUCT& cs)
33  {                                   //消息处理函数的实现
34      return CView::PreCreateWindow(cs);
35  }
36
37  ///////////////////////////////////////////////////////////////////
38  //CMouseTraceView 的绘画函数
39
40  void CMouseTraceView::OnDraw(CDC* pDC)
41  {                                   //绘图函数的实现
42      CMouseTraceDoc* pDoc = GetDocument();
43      ASSERT_VALID(pDoc);
44      pDC->TextOut(50, 100, m_showMsg);
45  }
46
47  #ifdef _DEBUG
48  void CMouseTraceView::AssertValid() const
49  {
50      CView::AssertValid();
51  }
52
53  void CMouseTraceView::Dump(CDumpContext& dc) const
54  {
55      CView::Dump(dc);
```

```
56  }
57
58  CMouseTraceDoc* CMouseTraceView::GetDocument()
59  {                                    //得到当前文档指针
60      ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMouseTraceDoc)));
61      return (CMouseTraceDoc*)m_pDocument;
62  }
63  #endif //_DEBUG
64
65  void CMouseTraceView::OnMouseMove(UINT nFlags, CPoint point)
66  {
67      m_showMsg.Format("x=%d, y=%d", point.x, point.y);
                                         //把鼠标的坐标保存转换成字符串
68      Invalidate();                    //使窗口失效，能自动更新窗口
69  }
```

代码解析：第 44 行，通过 pDC 指针指向的 TextOut()函数，在指定坐标位置显示相应的字符串信息。第 67 行，在响应鼠标移动消息时，将得到的鼠标的坐标转换成字符串并保存到 m_showMsg 变量之中。

（11）保存并编译执行程序。其效果如图 5.27 所示。

至此，整个示例程序的设计已经完成。从最后的效果图中可以看到，在主界面的窗口中已经把当前鼠标坐标位置显示出来，起到了“鼠标追踪”的作用。

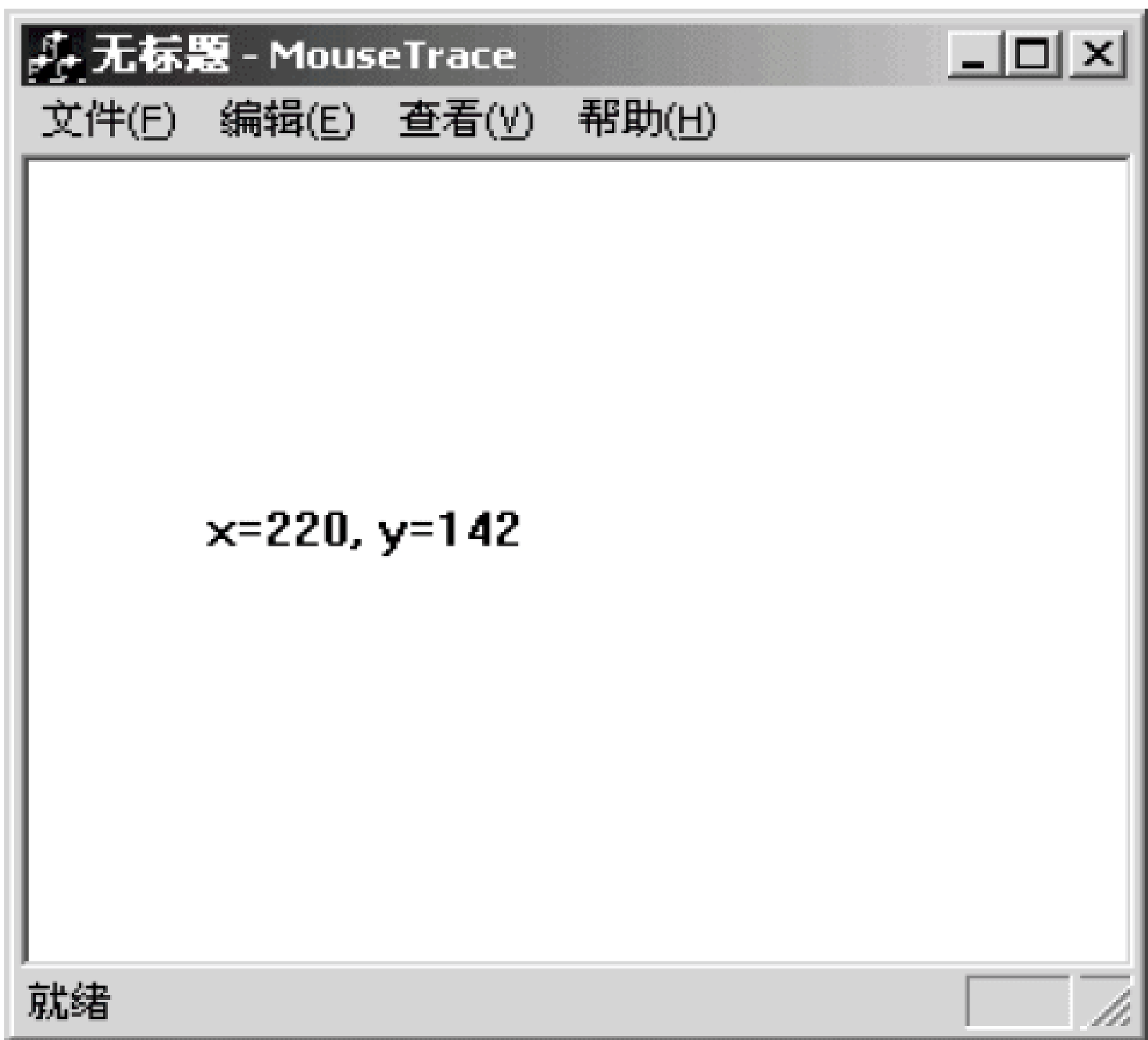


图 5.27　程序执行效果

5.6　两个入门小实例

前面讲解的示例都是比较简单的功能演示程序，在实际应用中的用处并不是太大。在这里为了增加各位读者对实用应用程序开发的经验，本节中将给出两个真实的应用开发示例。

5.6.1　简单的 MP3 播放器

相信很多读者都使用过电脑来欣赏美妙动听的音乐，而在电脑上最常用的音乐格式就是 MP3 格式的。电脑上有很多的播放软件支持这种格式，例如 winamp、千千静听、kugoo 等。不过这些软件都是由其他程序员设计的。不知道读者们有没有兴趣设计和编写一个属于自己的 MP3 播放软件呢？如果有兴趣，下面就跟笔者一起来编写吧！

1．MCI 接口介绍

因为 MP3 格式的音乐文件，是采用压缩算法来保存的音频文件。所以要播放这种音频文件，就必须知道 MP3 文件的解码方法，并对其进行解码转换成音频流格式，最后才能播

放出来。不过为了开发和讲解方便，笔者直接调用了操作系统内置的 MP3 解码引擎，用其来实现 MP3 音乐的播放。

要调用内置的 MP3 解码引擎，就必须先来了解一下 MCI（Media Control Interface）媒体控制接口。MCI 是微软公司提供的一组多媒体设备和文件的标准接口，其优点是可以方便地控制绝大多数多媒体设备，包括音频、视频、影碟、录像等多媒体设备，而不需要知道它们的内部工作状况。

接口所支持的媒体格式包括 avi、wav、mpeg、MP3、wma 等。笔者所要做的就是利用这公开的访问接口直接调用系统内置引擎实现 MP3 的播放。接口函数的声明如下：

```
MCIERROR WINAPI mciSendCommand (MCIDEVICEID mciId,
                                UINT uMsg,
                                DWORD dwParam1,
                                DWORD dwParam2)
```

其中，mciTd 指定了设备标识，这个标识会在程序员打开 MCI 设备时由系统提供。uMsg 指定将如何控制设备，即 MCI 指令，比较常用的指令如表 5.2 所示。最后两个参数一般是一个数据结构，标识程序在访问 MCI 时要的信息。

表 5.2　MCI常用指令

指　令　宏	说　　明
MCI_OPEN	打开文件，得到 MCI 接口设备
MCI_CLOSE	关闭文件，释放 MCI 接口设备
MCI_PLAY	播放文件
MCI_STOP	停止播放
MCI_PAUSE	暂停播放
MCI_STATUS	得到当前状态

2. 制作流程

下面笔者就来一步步地创建 MP3 播放器应用程序——Mp3Player。

（1）创建一个基于对话框模式的应用程序，界面上需要放置 4 个按钮和 1 个进度条，其 ID 和名称参见表 5.3 所示。

表 5.3　MP3 播放器界面ID及名称

ID	名　　称	ID	名　　称
IDC_OPEN_BTN	打开文件	IDC_STOP_BTN	停止
IDC_PLAY_BTN	播放	IDC_PROCESS	进度条
IDC_PAUSE_BTN	暂停		

技巧：增加滑动条来显示进度可以有效地提高用户的友好感。

各控件放置位置及主界面如图 5.28 所示。

（2）新建一个 CMyPlayerControl 类，其类声明放在 MyPlayerControl.h 文件中。注意要把 mmsystem.h 头文件加入。其代码如代码 5.11 所示。



图 5.28　Mp3Player 主界面

代码 5.11　CMyPlayerControl 类的声明

```
01  #ifndef __MY_PLAYER_CONTROL_H__
02  #define __MY_PLAYER_CONTROL_H__
03
04  #include <mmsystem.h>                       //插入多媒体头文件
05
06  class CMyPlayerControl
07  {
08  public:
09      CMyPlayerControl();                     //构造函数
10      ~CMyPlayerControl();                    //析构函数
11  public:
12      BOOL Open(LPCSTR lpFileName);           //打开文件
13      void Play();                            //播放
14      void Close();                           //关闭
15      void Stop();                            //停止
16      void Pause();                           //暂停
17      DWORD GetLength(DWORD dwItem);          //得到歌曲长度
18      void SetWindowsHwnd(HWND hWnd);         //设置主窗口句柄
19  private:
20      MCI_OPEN_PARMS mciOpen;                 //打开设备参数
21      HWND m_hWnd;                            //主窗口句柄
22      DWORD dwFrom;                           //播放起始点
23  };
24
25  #endif
```

（3）类声明编写完成后，就需要实现这个类，实现放在 MyPlayerControl.cpp 文件中，如代码 5.12 所示。要注意，因为类中调用了系统的 MCI 接口的 API 函数，所以必须加入 winmm.lib 这个静态库文件。

代码 5.12　CMyPlayerControl 类的实现

```
01  #include "stdafx.h"
02  #include "MyPlayerControl.h"                //插入类的声明头文件
03
04  CMyPlayerControl::CMyPlayerControl()        //构造函数
05  {
06  }
07
08  CMyPlayerControl::~CMyPlayerControl()       //析构函数
09  {
10      Close();                                //调用关闭
11  }
12
13  DWORD CMyPlayerControl::GetLength(DWORD dwItem)
14  {
15      //得到当前文件状态
```

```
16      MCI_STATUS_PARMS mcistatusparms;
17      mcistatusparms.dwCallback=(DWORD)m_hWnd;
18      mcistatusparms.dwItem=dwItem;          //状态类别值
19      mcistatusparms.dwReturn=0;
20      mciSendCommand(mciOpen.wDeviceID,
                      MCI_STATUS,
                      MCI_STATUS_ITEM,
                      (DWORD)&mcistatusparms);
21      return mcistatusparms.dwReturn;      //返回长度
22  }
23
24  BOOL CMyPlayerControl::Open(LPCSTR lpFileName)
25  {
26      //如果有打开的 MCI 设备就关闭
27      if (mciOpen.wDeviceID) Close();
28      //初始化 MCI_OPEN_PARMS 结构中的文件类型
29      mciOpen.lpstrDeviceType=NULL;
30      //播放文件路径
31      mciOpen.lpstrElementName=lpFileName;
32      //向 MCI 设备发送命令消息（在打开设备时，设备号为 0）
33      if ( mciSendCommand(0,
                      MCI_OPEN,
                      MCI_DEVTYPE_WAVEFORM_AUDIO,
        (DWORD)&mciOpen))
34      {
35          return FALSE;
36      }
37      dwFrom = MCI_MAKE_HMS(0,0,0);         //起始位置为 0
37      return TRUE;
38  }
39
40  void CMyPlayerControl::Play()
41  {
        //播放参数结构
        MCI_PLAY_PARMS mciplayparms;
        //得到文件大小
43      DWORD cdlen = GetLength(MCI_STATUS_LENGTH);
44      DWORD cdto=MCI_MAKE_HMS(MCI_HMS_HOUR(cdlen),
                MCI_HMS_MINUTE(cdlen),
                MCI_HMS_SECOND(cdlen));      //把文件中读出的大小转换为时间数量
45      mciplayparms.dwCallback=NULL;
46      mciplayparms.dwFrom=dwFrom;           //设置起始位置
47      mciplayparms.dwTo=cdto;               //设置终止位置
48      if(mciOpen.wDeviceID != 0)           //判断是否打开文件
        {//播放音乐
            mciSendCommand(mciOpen.wDeviceID,
                          MCI_PLAY,
                          MCI_TO|MCI_FROM,
                          (DWORD)(LPVOID)& mciplayparms);
        }
49  }
50
51  void CMyPlayerControl::Close()
52  {
53      if(mciOpen.wDeviceID)
54      {
55      //执行 MCI_CLOSE 操作，关闭 MCI 设备
56          mciSendCommand(mciOpen.wDeviceID,MCI_CLOSE,NULL,NULL);
57      }
```

```
58  }
59
60  void CMyPlayerControl::Stop()
61  {
62      if (mciOpen.wDeviceID)
63      {//执行 MCI_STOP 操作，停止播放音乐
64          mciSendCommand(mciOpen.wDeviceID,MCI_STOP,NULL,NULL);
65      //把播放位置设定为音乐文件的开头（使下一次播放操作从文件开头位置开始）
66      mciSendCommand(mciOpen.wDeviceID,
                        MCI_SEEK,
                        MCI_SEEK_TO_START,
                        NULL);
67      }

68      dwFrom = MCI_MAKE_HMS(0,0,0);       //把起始位置设置为 0
69  }
70
71  void CMyPlayerControl::Pause()
72  {
73      if (mciOpen.wDeviceID)
74      {//执行 MCI_PAUSE 操作，暂停播放音乐
75          DWORD dwsf=GetLength(MCI_STATUS_POSITION);
76          dwFrom=MCI_MAKE_MSF(MCI_MSF_MINUTE(dwsf),
77                              MCI_MSF_SECOND(dwsf),
78                              MCI_MSF_FRAME(dwsf));
79      //执行 MCI_PAUSE 操作，暂停播放音乐
80          mciSendCommand(mciOpen.wDeviceID,MCI_PAUSE,NULL,NULL);
81
82      }
83
84  }
85
86  void CMyPlayerControl::SetWindowsHwnd(HWND hWnd)
87  {
88      m_hWnd = hWnd;                      //把当前父窗口的句柄传入
89  }
```

技巧： 使用类的定义方法，可以使这个类重用性更高。

代码解析：代码中第 33、48、56、64 及 80 行是调用 mciSendCommand()函数来实现对音频文件的打开、关闭、播放、暂停、停止等操作，只是通过不同的参数进行控制。例如，第 33 行代码，其中参数 MCI_OPEN 是打开操作。第 48 行代码，参数 MCI_PLAY 是播放操作。第 56 行代码，其中参数 MCI_CLOSE 是关闭操作。第 80 行代码，参数 MCI_PAUSE 是暂停操作。

（4）至此，这个 CMyPlayerControl 类已经实现。现在要在对话框类的声明中，加入相应的头文件，并定义一个 CMyPlayerControl 类的对象，同时通过类向导为 CMp3PlayerDlg 类添加各按钮响应函数，其代码如代码 5.13 所示。

代码 5.13　CMp3PlayerDlg 类的声明

```
01  //Mp3PlayerDlg.h : 头文件
02
03
04  #if !defined(AFX_MP3PLAYERDLG_H__)
05  #define AFX_MP3PLAYERDLG_H__
```

```
06
07  #if _MSC_VER > 1000
08  #pragma once
09  #endif //_MSC_VER > 1000
10
11  ///////////////////////////////////////////////////////////////////
12  //CMp3PlayerDlg 对话框
13  #include "MyPlayerControl.h"    //插入 CMyPlayerControl 类的声明头文件
14
15  class CMp3PlayerDlg : public CDialog       //继承于 CDialog 类
16  {
17  public:
        //构造函数
18      CMp3PlayerDlg(CWnd* pParent = NULL);
19      //对话框的数据
20      //{{AFX_DATA(CMp3PlayerDlg)
21      enum { IDD = IDD_MP3PLAYER_DIALOG };
22      CSliderCtrl m_process;                 //滑动条对象
23      //}}AFX_DATA
24
25      //类中各个虚函数声明
26      //{{AFX_VIRTUAL(CMp3PlayerDlg)
27      protected:
28      //对话框资源加载函数声明
29      virtual void DoDataExchange(CDataExchange* pDX)
30      //}}AFX_VIRTUAL
31  private:
32      CMyPlayerControl m_myPlayerControl;    //类对象
33      CString m_strFileName;                 //打开文件路径
34  protected:
35      HICON m_hIcon;
36
37  //消息映射函数声明
38      //{{AFX_MSG(CMp3PlayerDlg)
39      virtual BOOL OnInitDialog();           //初始化对话框函数
40      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
41      afx_msg void OnPaint();                //绘图函数
42      afx_msg HCURSOR OnQueryDragIcon();
43      virtual void OnOK();                   //单击回车键响应
44      virtual void OnCancel();               //按下 Esc 键时响应，退出
45      afx_msg void OnOpenBtn();              //打开按钮响应
46      afx_msg void OnPauseBtn();             //暂停按钮响应
47      afx_msg void OnPlayBtn();              //播放按钮响应
48      afx_msg void OnStopBtn();              //停止按钮响应
49      //}}AFX_MSG
50      DECLARE_MESSAGE_MAP()
51  };
52
53  #endif //!defined(AFX_MP3PLAYERDLG_H)
```

代码解析：代码第 45～48 行，是分别对打开、暂停、播放和停止按钮响应函数的声明。

（5）声明函数之后，就需要对这些按钮响应函数的功能进行实现。主要是调用 m_myPlayerControl 对象的成员函数及辅助设计，其代码如代码 5.14 所示。

代码 5.14　CMp3PlayerDlg 类的实现

```
01  #include "StdAfx.h"
02  #include "Mp3Player.h"
03  #include "Mp3PlayerDlg.h"          //插入类声明头文件
...                                    //省略部分代码，请查看光盘中的源代码
                                       //消息与成员函数映射
04  BEGIN_MESSAGE_MAP(CMp3PlayerDlg, CDialog)
05      ON_WM_SYSCOMMAND()
06      ON_WM_PAINT()
07      ON_WM_QUERYDRAGICON().
08                                     //打开按钮的响应函数为 OnOpenBtn
09      ON_BN_CLICKED(IDC_OPEN_BTN, OnOpenBtn)
10                                     //暂停按钮的响应函数为 OnPauseBtn
11      ON_BN_CLICKED(IDC_PAUSE_BTN, OnPauseBtn)
12                                     //播放按钮的响应函数为 OnPlayBtn
13      ON_BN_CLICKED(IDC_PLAY_BTN, OnPlayBtn)
14                                     //停止按钮的响应函数为 OnStopBtn
15      ON_BN_CLICKED(IDC_STOP_BTN, OnStopBtn)
16      ON_WM_TIMER()
17  END_MESSAGE_MAP()
18
19  void CMp3PlayerDlg::OnPaint()      //绘图函数
20  {
21      if (IsIconic())
22      {
23          CPaintDC dc(this);         //得到绘图设备句柄
24
25          SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
26
27                                     //得到图标在客户区中的坐标
28          int cxIcon = GetSystemMetrics(SM_CXICON);
29          int cyIcon = GetSystemMetrics(SM_CYICON);
30          CRect rect;
31          GetClientRect(&rect);      //得到客户区的矩形范围
32          int x = (rect.Width() - cxIcon + 1) / 2;
33          int y = (rect.Height() - cyIcon + 1) / 2;
34
35          dc.DrawIcon(x, y, m_hIcon);//画出图标
36
37      }
38      else
39      {
40          CDialog::OnPaint();        //调用默认绘图
41      }
42  }
43
44  void CMp3PlayerDlg::OnOK()
45  {                                  //当按下 Enter 键时响应
46      CDialog::OnOK();
47  }
48
49  void CMp3PlayerDlg::OnCancel()
50  {                                  //当按下 Esc 键时响应，退出
51      CDialog::OnCancel();
52  }
53
54  void CMp3PlayerDlg::OnOpenBtn()
```

```
55  {
56                                              //文件类型过滤字符串
57      CString strFilter("MP3 Files (*.mp3)|*.mp3||");
58      CFileDialog  OpenDlg(TRUE,              //生成打开对话框
59                          NULL,
60                          NULL,
61                          OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
62                          strFilter);
63      int nFlags = OpenDlg.DoModal();         //通过打开对话框选择 MP3 音乐文件
64      if(nFlags==IDOK)
65      {//用户单击的是“打开”按钮，将文件路径名赋给成员变量 m_strFileName 备用
66          m_strFileName=OpenDlg.GetPathName();
67                                      //打开 MCI 设备，并将设备分配给选中的文件使用
68          m_myPlayerControl.Open(m_strFileName);
69      }
70  }
71
72  void CMp3PlayerDlg::OnPauseBtn()
73  {
74      m_myPlayerControl.Pause();              //调用暂停
75  }
76
77  void CMp3PlayerDlg::OnPlayBtn()
78  {
79                                              //得到曲目长度
80      DWORD cdlen = m_myPlayerControl.GetLength(MCI_STATUS_LENGTH);
81      m_myPlayerControl.Play();               //调用播放
82      m_process.SetRange(0, cdlen);           //设置进度条范围
83      SetTimer(0,1000,NULL);                  //设置定时器
84  }
85
86  void CMp3PlayerDlg::OnStopBtn()
87  {
88      m_myPlayerControl.Stop();               //调用停止
89      m_process.SetPos(0);                    //设置进度条的位置
90      KillTimer(0);                           //关闭定时器
91  }
92
93  void CMp3PlayerDlg::OnTimer(UINT nIDEvent)
94  {
95      //每秒钟得到的当前播放进度
96      DWORD cdf=m_myPlayerControl.GetLength(MCI_STATUS_POSITION);
97      m_process.SetPos(cdf);                  //设置进度条位置
98      CDialog::OnTimer(nIDEvent);             //调用其他 Timer
99  }
```

代码解析：代码第 58 行是创建 CFileDialog 对象，然后第 68 行调用 m_myPlayerControl 对象的 open()函数打开指定音频文件，方便后面的操作。

（6）测试程序效果。操作方法为：单击“打开”按钮，弹出“打开”对话框。指定文件，并单击对话框中的“打开”按钮，如图 5.29 所示。

（7）打开指定的文件后，单击“播放”按钮。这时进度条会自动向右增加，并且可以听到音乐开始播放了，其效果如图 5.30 所示。

技巧： 多使用系统自带的对话框对象，比自己去创建更友好，效率更高。

这个例子只支持 MP3 格式的音乐，读者可以自己参阅 MSDN，试试通过对 MCI 接口的编程，实现对多种格式的音视频文件的播放控制，具体的实现方法与上例类似。



图 5.29　Mp3Player 程序效果——打开文件



图 5.30　Mp3Player 程序效果——播放文件中

5.6.2　简单的图片浏览器

前面介绍图片显示时，其中各种显示图片的方法，只支持 BMP 格式的图像文件。在现实中，这是无法满足要求的。所以在这一节中，笔者将设计一个利用 Internet Explorer 为后台支持的，能够浏览多种格式的自制图片浏览器。其中需要用到 CHtmlView 类。

1. CHtmlView 类介绍

该类是微软公司在 MFC 的 Document/View 环境中封装了 WebBrowse 控件的功能。通过这个类提供的函数，用户可以很方便地建立自己的浏览器。这样就可以实现浏览网站、本地和网络的资源。

Internet Explorer 的绝大部分功能，CHtmlView 类都已经包含。因为 WebBrowse 控件实际上是基于 IWebBrowser2 接口的，而该接口是由 Internet Explorer 实现的一个 COM 接口。因此，CHtmlView 类实际上使用的就是 Internet Explorer 引擎，故而具有 Internet Explorer 的能力。

笔者就是利用这个类，通过它能浏览网页和图片的功能。把本地图片文件的路径写入并链接到一个临时的 HTML 网页文件中，再通过该类的接口函数 Navigate2 浏览这个网页文件，就可以实现把各种类型的图片显示出来的要求。

该接口函数 Navigate2()的原型如下：

```
void Navigate2(LPCTSTR lpszURL,
            DWORD dwFlags = 0,
            LPCTSTR lpszTargetFrameName = NULL,
            LPCTSTR lpszHeaders = NULL,
            LPVOID lpvPostData = NULL,
            DWORD dwPostDataLen = 0
);
```

用户只需要指定其中的 lpszURL 参数，其是指浏览资源路径。其他的几个都使用默认值即可。

2．制作流程

下面笔者就来一步步地创建图片浏览器应用程序——PicBrowser。过程如下所述。

（1）创建一个基于框架结构的应用程序，其中需要在向导中设置框架结构为 SDI 单文档界面，并且在第 6 步中把 View 的基类设置为 ChtmlView，如图 5.31 所示。

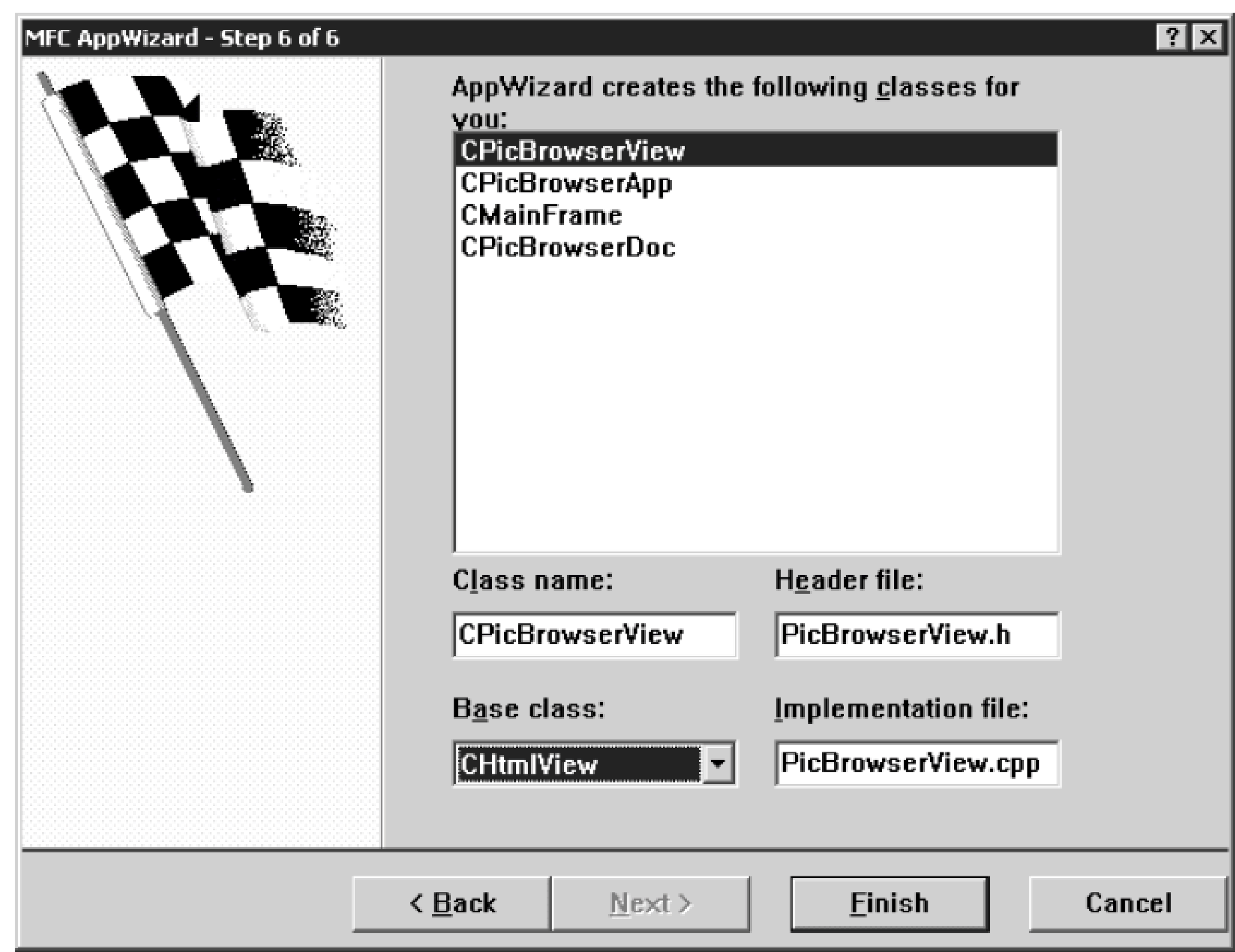


图 5.31　把 View 的基类设置为 CHtmlView

（2）创建一个 HTML 文件格式的字符串，如下所示。

```
"<html>"\
"<head>"\
"<TITLE>图片浏览器</TITLE>"\
"</head>"\
"<body>"\
"<DIV id=\"image\" class=\"slides\"><IMG src=\"pic/1.jpg\"></DIV>"\
"</body>"\
"</html>";
```

这段字符串生成的 HTML 文件内容如下：

```
<html>
<head>
<TITLE>图片浏览器</TITLE>
</head>
<body >
<DIV id="image" class="slides"><IMG src="pic/1.jpg"></DIV>
</body>
</html>
```

（3）给 CPicBrowserView 增加一个可以修改字符串中文件路径的设置函数 SetFile()，这个函数的参数为指定文件的路径，其代码如代码 5.15 所示。

代码 5.15　SetFile()函数实现

```
01  void CPicBrowserView::SetFile(LPCSTR lpFileName)
02  {
03      CString tmp;
04      FILE * fp = NULL;                   //文件指针
```

```
05      CString m_TempFile;                     //临时文件路径
06      if(lpFileName != NULL)                  //如果路径为空，不设置
07      {
08          tmp.Format("<html>"\
09                      "<head>"\
10                      "<TITLE>图片浏览器</TITLE>"\
11                      "</head>"\
12                      "<body>"\
13                      "<DIV id=\"image\" class=\"slides\"><IMG src=
                        \"%s\"></DIV>"\
14                      "</body>"\
15                      "</html>", lpFileName);
16          m_TempFile = GetExeFilePath();
17          m_TempFile+="\\temp.html";
18       //打开文件，如果文件存在，则删除，否则就创建
19          fp = fopen(m_TempFile, "w+");
20       //写入文件
21          int nSize = fprintf(fp, "%s", tmp);
22       //关闭文件
23          fclose(fp);
24      }
25  }
```

（4）给工具栏增加一个浏览按钮，并设置其 ID 为 ID_BROWER_BTN，界面如图 5.32 所示。

（5）使用类向导给工具栏中的浏览按钮增加响应函数（OnBrowerBtn），放置于 CPicBrowserView 类中，如图 5.33 所示。

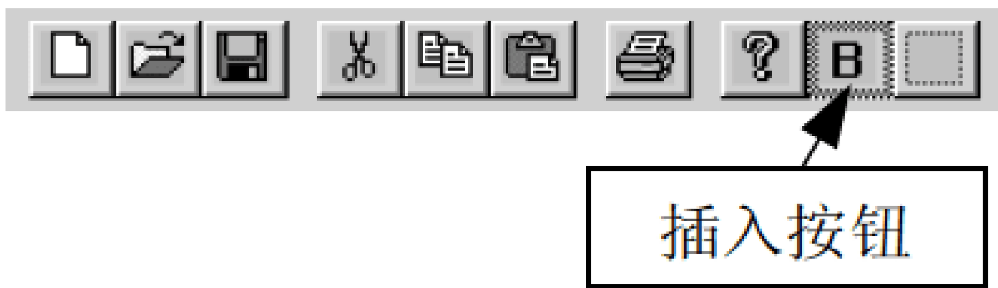


图 5.32　给工具栏浏览按钮

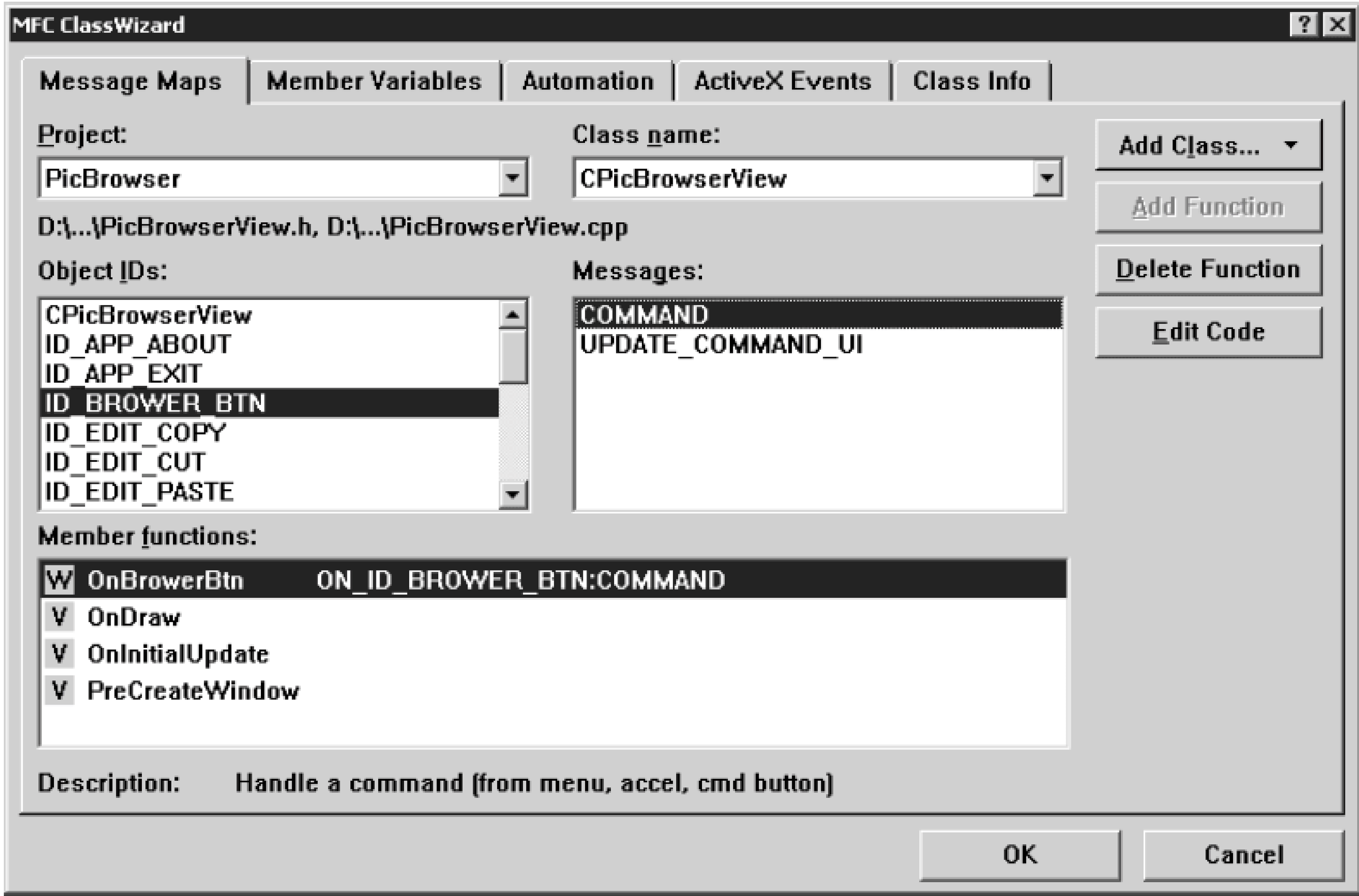


图 5.33　添加浏览按钮响应函数

（6）修改 CPicBrowserView 的声明文件，如代码 5.16 所示。

技巧：BitBlt 和图片控件只能显示 BMP 类的图片，而 WebBrower 控件则可以显示互联网上所有的图片资源。

代码 5.16　CPicBrowserView 类的声明

```
01  #if !defined(AFX_PICBROWSERVIEW_H__)
02  #define AFX_PICBROWSERVIEW_H__
```

```
03
04  class CPicBrowserView : public CHtmlView   //CPicBrowserView 类的定义
05  {
06  protected:
07      CPicBrowserView();                     //构造函数，由系统调用
08      DECLARE_DYNCREATE(CPicBrowserView)
09
10                                             //各种数据成员及函数的声明
11  public:
12      CPicBrowserDoc* GetDocument();
13
14                                             //在这里定义各种操作接口函数
15  public:
16      virtual void OnDraw(CDC* pDC);         //重载的绘图函数在本视图中
17      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
18  protected:
19      virtual void OnInitialUpdate();   //在构造函数后，自动调用的初始化函数
20
21  public:                                    //自定义的成员函数
22      CString GetExeFilePath();              //得到执行文件的全路径
23      void SetFile(LPCSTR lpFileName);       //调用文件路径
24      virtual ~CPicBrowserView();            //析构函数
25  #ifdef _DEBUG
26      virtual void AssertValid() const;
27      virtual void Dump(CDumpContext& dc) const;
28  #endif
29
30  protected:
31
32                                             //各种消息与函数的映射
33  protected:
34      afx_msg void OnBrowerBtn();            //单击浏览按钮时响应
35      DECLARE_MESSAGE_MAP()
36  };
```

（7）修改 picBrowserView 类的实现如代码 5.17 所示。

代码 5.17　picBrowserView 类的实现

```
01  #include "stdafx.h"
02  #include "PicBrowser.h"
03
04  #include "PicBrowserDoc.h"
05  #include "PicBrowserView.h"                //插入类的声明头文件
06
07  IMPLEMENT_DYNCREATE(CPicBrowserView, CHtmlView)
08
09  BEGIN_MESSAGE_MAP(CPicBrowserView, CHtmlView)
10  //把单击 ID_BROWER_BTN 按钮资源与函数 OnBrowrBtn 映射
11      ON_COMMAND(ID_BROWER_BTN, OnBrowerBtn)
12  //打印按钮与函数的响应
13      ON_COMMAND(ID_FILE_PRINT, CHtmlView::OnFilePrint)
14  END_MESSAGE_MAP()
15
16  CPicBrowserView::CPicBrowserView()         //构造函数
17  {
18  }
19
```

```
20  CPicBrowserView::~CPicBrowserView()        //析构函数
21  {
22  }
23
24  BOOL CPicBrowserView::PreCreateWindow(CREATESTRUCT& cs)
25  {                                           //创建窗口前的预处理
26      return CHtmlView::PreCreateWindow(cs);
27  }
28
29  void CPicBrowserView::OnDraw(CDC* pDC)      //绘图实现
30  {
31      CPicBrowserDoc* pDoc = GetDocument();   //得到当前文档
32      ASSERT_VALID(pDoc);
33  }
34
35  void CPicBrowserView::OnInitialUpdate()
36  {
37      CHtmlView::OnInitialUpdate();           //初始化函数
38
39  }
40
41  ///////////////////////////////////////////////////////////////////////////
42  //CPicBrowserView 类的消息响应函数的实现
43
44  void CPicBrowserView::SetFile(LPCSTR lpFileName)
45  {
46      CString tmp;
47      FILE * fp = NULL;                       //文件指针初始化为空
48      CString m_TempFile;                     //用于保存文件路径字符串
49      if(lpFileName != NULL)
50      {                                       //如果路径为空，不设置
51          tmp.Format("<html>"\
52                      "<head>"\
53                      "<TITLE>图片浏览器</TITLE>"\
54                      "</head>"\
55                      "<body>"\
56                      "<DIV id=\"image\" class=\"slides\"><IMG
                        src=\"%s\"></DIV>"\
57                      "</body>"\
58                  "</html>", lpFileName);
59          m_TempFile = GetExeFilePath();
60          m_TempFile+="\\temp.html";
61                                  //打开文件，如果文件存在，则删除，否则就创建
62          fp = fopen(m_TempFile, "w+");
63                                      //写入文件
64          int nSize = fprintf(fp, "%s", tmp);
65
66          fclose(fp);                 //关闭文件
67      }
68  }
69
70  void CPicBrowserView::OnBrowerBtn()
71  {
72      CString strFileName;
73      CString tmpFilePath;
74                                      //文件类型过滤字符串，其以“||”结束
75      CString strFilter("JPEG 文件 (*.jpg)|*.JPG| 动画文件
        (*.gif)|*.GIF||");
```

```
76                                                  //生成打开对话框
77      CFileDialog  OpenDlg(TRUE,
78           NULL,
79           NULL,
80           OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
81           strFilter);
82
83      int nFlags = OpenDlg.DoModal();//通过打开对话框选择图片文件
84      if(nFlags==IDOK)
85      {//用户单击的是“打开”按钮，将文件路径名赋给成员变量 m_strFileName 备用
86           strFileName=OpenDlg.GetPathName();
87                                                  //设置文件路径，并保存文件
88           SetFile(strFileName);
89                                                  //得到想要浏览的临时 HTML 文件
90           strFileName = GetExeFilePath();
91           strFileName += "\\temp.html";
92           tmpFilePath.Format("file:///%s", strFileName);
93
94           Navigate2(tmpFilePath, NULL, NULL); //浏览临时 HTML 文件
95
96           Refresh();                                 //更新网页显示
97      }
98  }
99
100 CString CPicBrowserView::GetExeFilePath()
101 {
102     char exeFullPath[MAX_PATH];
103     CString m_TempFile;
104     GetModuleFileName(NULL,exeFullPath,MAX_PATH);
105                                                  //将其格式化为字符串
106     m_TempFile.Format("%s",exeFullPath);
107     //去掉应用程序的全名（15 为应用程序文件全名的长度）
108     exeFullPath[m_TempFile.GetLength()-15]='\0';
109                                                  //得到应用程序的所在路径
110     m_TempFile.Format("%s",exeFullPath);
111     //得到临时文件的全路径
112     return m_TempFile;
113 }
```

代码解析：代码第 44 行，SetFile()函数实现的核心就是将指定 html 文件的内容写入 temp.html 中保存。代码第 94 行，就是调用 Navigate2()函数浏览和显示刚才前面替换后的文件，完成图像文件显示功能。

（8）CPicBrowserApp 类的声明如代码 5.18 所示。

代码 5.18　CPicBrowserApp 类的声明

```
01  //PicBrowser.h : 主函数的头文件
02
03
04  #if !defined(AFX_PICBROWSER_H__)
05  #define AFX_PICBROWSER_H__
06
07  #if _MSC_VER > 1000
08  #pragma once
09  #endif //_MSC_VER > 1000
10
11  #ifndef __AFXWIN_H__
12      #error include 'stdafx.h' before including this file for PCH
```

```
13  #endif
14
15  #include "resource.h"                          //全局资源定义
16
17  ///////////////////////////////////////////////////////////////////
18  //CpicBrowserApp 类的声明
19
20
21
22  class CPicBrowserApp : public CWinApp
23  {
24  public:
25      CPicBrowserApp();                          //构造函数
26
27  //成员函数声明
28                                                 //使用类向导创建的重载的虚函数
29      //{{AFX_VIRTUAL(CPicBrowserApp)
30      public:
31      virtual BOOL InitInstance();               //初始化应用程序
32      //}}AFX_VIRTUAL
33
34  //消息函数声明
35      //{{AFX_MSG(CPicBrowserApp)
36      afx_msg void OnAppAbout();                 //关于按钮的响应
37      //}}AFX_MSG
38      DECLARE_MESSAGE_MAP()
39  };
40  #endif
```

（9）CPicBrowserApp 类的实现如代码 5.19 所示。

代码 5.19　CPicBrowserApp 类的实现

```
01  //MainFrm.cpp : implementation of the CMainFrame class
02
03
04  #include "stdafx.h"                        //插入头文件
05  #include "PicBrowser.h"
06
07  #include "MainFrm.h"                       //插入主框架类的头文件
08
09  #ifdef _DEBUG
10  #define new DEBUG_NEW
11  #undef THIS_FILE
12  static char THIS_FILE[] = __FILE__;
13  #endif
14
15  ///////////////////////////////////////////////////////////////////
16  //CMainFrame 的实现代码开始
18  IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
19                      //消息与成员函数的映射
20  BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
21      ON_WM_CREATE()
22  END_MESSAGE_MAP()
23
24  static UINT indicators[] =
25  {
26      ID_SEPARATOR,                      //状态栏 ID
27      ID_INDICATOR_CAPS,
```

```
28      ID_INDICATOR_NUM,
29      ID_INDICATOR_SCRL,
30  };
33
34  ///////////////////////////////////////////////////////////////////////
35  //CMainFrame 构造与析构函数
36
37  CMainFrame::CMainFrame()            //构造函数
38  {
39  }
40
41  CMainFrame::~CMainFrame()           //析构函数
42  {
43  }
44
45  int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
46  {                                   //创建函数
47      if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
48          return -1;
49                                      //创建工具栏并加载
50      if (!m_wndToolBar.CreateEx(this,
                          TBSTYLE_FLAT,
                          WS_CHILD | WS_VISIBLE | CBRS_TOP
                          |CBRS_GRIPPER|CBRS_TOOLTIPS|
                          CBRS_FLYBY|CBRS_SIZE_DYNAMIC) ||
                          !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
51      {
52          TRACE0("Failed to create toolbar\n");
53          return -1;                  //创建失败，返回-1
54      }
56      if (!m_wndStatusBar.Create(this) ||
57          !m_wndStatusBar.SetIndicators(indicators,
58            sizeof(indicators)/sizeof(UINT)))
59      {
60          TRACE0("Failed to create status bar\n");
61          return -1;                  //创建失败，返回-1
62      }
63                                      //工具栏有效
64      m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
65      EnableDocking(CBRS_ALIGN_ANY);
66      DockControlBar(&m_wndToolBar);
67
68      return 0;                       //返回 0，表示成功
69  }
70
71  BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
72  {                                   //创建窗口前的预处理
73      if( !CFrameWnd::PreCreateWindow(cs) )
74          return FALSE;
75      return TRUE;
76  }
```

（10）至此，整个程序代码编写完毕。编译并执行程序，最后的效果如图 5.34 所示。

笔者现在演示的只是浏览 JPG 和 GIF 的文件，读者可以自己编写程序，尝试让这个图片浏览器支持更多的格式。

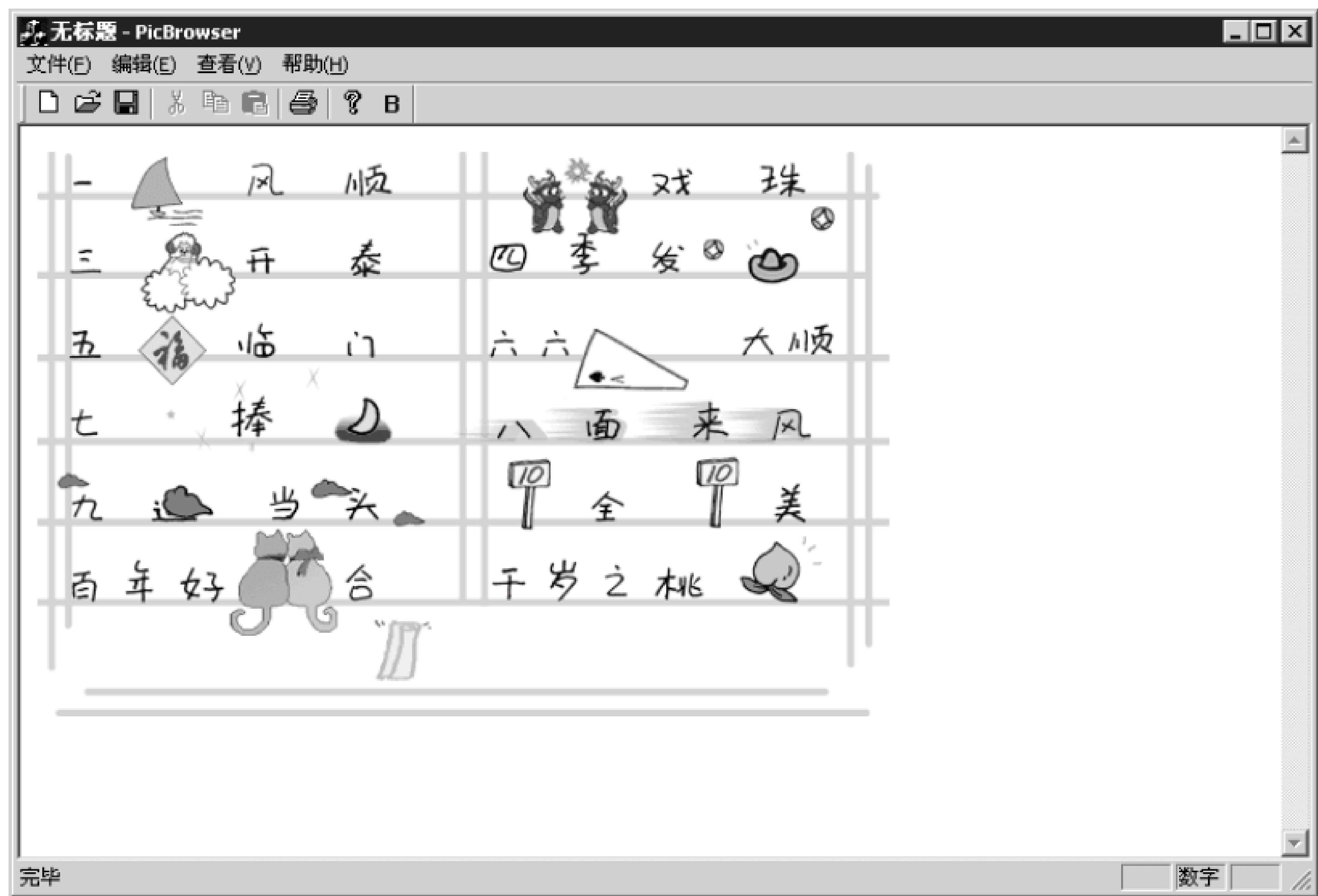


图 5.34　PicBrowser 的程序执行效果

5.7　总　　结

在本章的学习中，相信各位读者已经知道什么是多媒体及其特点，并且掌握了 Windows 中的各种类型的图像及 Visual C++中如何显示这些图像的办法。同时对于声音和音乐的播放也已经掌握了。

最后通过两个开发实例的实践，相信读者已经可以自己设计出漂亮、实用的 Windows 应用程序了。这也是进入游戏开发的必经之路。在第 6 章中，读者将会了解和掌握项目管理的相关内容。

5.8　挑　　战

一、问答题

1．传统的多媒体是什么？其特点有哪些？

2．Windows 系统上的多媒体文件包含哪些类别？

3．图像文件中，BMP 格式的文件有哪些特点？

4．Visual C++中可以使用几种方法显示图像？这几种文件分别是什么？

5．系统对于输入设备的处理方法是什么？其优点是什么？

二、编程题

根据下列要求开发一个简单的多媒体播放器。

1．能够播放声音和视频文件。

2．支持各种控制，例如播放、暂停、停止等操作。

3．可以支持播放列表。

第 6 章　项目管理基础

游戏开发中由于涉及的各种内容比较多，例如文档、代码、图像、声音及用户与程序员等，为了使一款游戏最后能够被成功的设计和开发出来，就需要建立一个项目，并对其进行管理。本章就来讲解项目管理的基础内容。

本章主要涉及的内容如下：

- 项目及其管理的概念：知道什么是项目；知道什么是项目管理及其特点。
- 需求分析：把用户的语言描述转换成各种游戏开发的文档。
- 项目计划安排：在对需求进行分析后，如何制定开发计划及保证项目的成功。
- 各种设计文档：了解项目开发中需要涉及的文档及其格式。

6.1　项 目 管 理

在学习游戏项目管理之前，读者肯定会有疑问：什么是项目？项目管理又是什么？其特点和优势有哪些？下面笔者将逐一介绍。

6.1.1　项目与项目管理概念

项目指一系列独特的、复杂的并相互关联的活动。这些活动有着一个明确的目标或目的，必须在特定的时间、预算、资源限定内，依据规范完成。项目的要素包括范围、质量、成本、时间、资源。例如，举办一次展览会、开发一种新产品、建一个新小区、企业制定一个新战略等都可以称之为一个项目。

项目管理是基于科学管理原则的一套技术管理方法，这些技术或方法用于项目的计划、有效性评估、控制项目开发进度工作等活动，以按时、按预算、依据规范达到理想的最终效果。

6.1.2　项目管理的特点

项目管理工作具有以下几个特点：

（1）一次性。所谓项目的一次性是指项目与其他重复性运行或操作工作最大的区别。项目有明确的起点和终点，没有可以完全照搬的先例，也不会有完全相同的复制。项目的其他特点也是从这一主要的特点衍生出来的。

（2）独特性。每个项目都是独特的，或者其提供的产品或服务有自身的特点；或者其提供的产品或服务与其他项目类似，然而其时间和地点、内部和外部的环境、自然和社会

条件有别于其他项目，因此项目的过程总是独一无二的。

（3）目标的确定性。项目必须有确定的目标，例如：

- 时间性目标，如在规定的时段内或规定的时点之前完成。
- 成果性目标，如提供某种规定的产品或服务。
- 约束性目标，如不超过规定的资源限制。
- 其他需满足的要求，包括必须满足的要求和尽量满足的要求。

目标的确定性允许有一个变动的幅度，也就是可以修改。一旦项目目标发生实质性变化，其就不再是原来的项目了，而将产生一个新的项目。

（4）活动的整体性。项目中的一切活动都是相关联的，构成一个整体。多余的活动是不必要的，缺少某些活动必将损害项目目标的实现。例如，如果在早期需求不明确的情况下，就开始项目开发，那么最终开发出来的产品一定是不能满足需求的。

（5）组织的临时性和开放性。项目开发团队在项目的全过程中，其人数、成员、职责都是在不断变化的。例如，某些项目开发团队的成员是借调来的，项目终结时开发团队就要解散，开发人员就要转移。而且如果是一个大型项目，参与项目的团队往往有多个。各团队通过协议或合同以及其他的社会关系组织到一起，在项目的不同时段不同程度地介入项目开发活动。可以说，项目组织没有严格的边界，是临时性、开放性的。这一点与一般企、事业单位和政府机构组织很不一样。

（6）成果的不可挽回性。项目的一次性特点决定了项目不同于其他事情可以试做，做坏了可以重来；也不同于生产批量产品，合格率达 99.99%就是很好的了。项目在一定条件下启动，一旦失败就永远失去了重新进行原项目的机会。项目相对于运作有较大的不确定性和风险。

综合以上特点，项目管理工作的目的是在项目活动中运用科学知识、技能、工具和技术，以满足和超过项目相关人对项目的需求和期望。

6.1.3　采用项目管理的优势

当设定一个项目后，按照传统的做法，参与这个项目的至少会有很多个不同的组织和个人，而不同组织和个人在运作项目过程中不可避免地会产生摩擦，需进行各种协调，而这些无疑会增加整个项目的成本，影响项目实施的效率。

而采用项目管理的做法，则效果会大大的不同。不同组织和个人因为某一个项目而组成一个团队，项目经理是整个项目团队的领导者，其所肩负的责任就是领导团队准时、优质地完成全部工作，在不超出预算的情况下实现项目目标。

项目的管理者不仅仅是项目执行者，其参与项目的需求确定、项目选择、计划直至收尾的全过程，并在时间、成本、质量、风险、合同、设计、测试、人力资源等各个方面对项目进行全方位的管理，因此项目管理可以解决需要跨领域或者人员沟通等复杂问题，并实现更高的运营效率。

项目管理是全新的管理方法，学习项目管理可以开阔思路和视野，能培养系统思维习惯，务实的工作作风，科学的管理方法；并养成良好的工作方式。

采用项目管理大致有如下几个优点：

（1）可以合理安排项目的整体进度，有效使用各项目资源，确保项目能够按期完成，

并降低项目成本。通过一系列项目管理方法和技术的使用，可以尽早地制定出项目的任务组成，并合理安排各项任务的先后顺序，有效安排资源的使用，特别是项目中的关键资源和重点资源，从而保证项目的顺利实施，并有效降低项目成本。如果不采用项目管理的方法，通常会盲目地启动一个项目，将所有资源均安排在项目中，这样可能会有很多的人员造成任务的瓶颈，同时也会造成很多的资源闲置，这样势必会造成资源和时间的浪费。

（2）加强项目的团队合作，提高项目团队的战斗力。项目管理的方法提供了一系列的人力资源管理、沟通管理的方法。通过这些方法的使用，可以增强团队合作精神，提高项目组成员的工作士气和效率。

（3）降低整个项目失败风险，提高项目实施的成功率。项目管理中重要的一部分是风险管理，通过风险管理可以有效降低项目的不确定因素对项目的影响。其实，这些工作是在传统的项目实施过程中最容易被忽略的，也是会对项目产生毁灭性后果的因素之一。

（4）有效控制项目范围，增强项目的可控性。在项目实施过程中，需求的变更是经常发生的。如果没有一种好的方法进行控制，势必会对项目产生很多不良的影响，而项目管理中强调进行范围控制，能有效降低项目范围变更对项目的影响，保证项目顺利实施。

（5）可以尽早地发现项目实施中的问题，有效地进行项目控制。对项目计划、执行状况的检查，能够及早地发现项目实施中存在的问题和隐含的问题，这样项目就能顺利执行。

（6）可以有效地进行项目的知识积累。传统的项目实施中，经常在项目实施完成时，项目就戛然而止，对于项目的实施总结、技术积累，都是一种空谈。项目管理中强调项目结束时，需要进行项目总结，这样就能将更多的项目经验，转换为项目财富。总体来讲，项目管理可以使得项目顺利实施，降低项目的风险性，最大限度地达到预期的目标。

6.2 软件工程与项目管理

为了解决电脑软件开发和维护过程中所遇到的一系列严重问题，软件也需要科学的项目管理方法。但这种项目管理方法只是软件工程的一部分。软件工程还包含很多内容，下面笔者将详细介绍软件工程的概念、重要性及其流程。

6.2.1 软件工程的概念

软件工程是一类解决软件设计和维护过程中遇到问题的工程。其应用计算机科学、数学及管理科学等原理，借鉴传统工程的原则、方法，创建软件以达到提高质量、降低成本的目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计范型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。软件工程是一门指导计算机软件开发和维护的工程学科。

软件工程准则可以概括为下列 6 条基本原理：

- 用分阶段的生存周期计划严格管理。
- 坚持进行阶段评审。
- 实行严格的产品控制。
- 采用现代程序设计技术。

- 应能清楚地审查结果。
- 合理安排软件开发小组的人员。

6.2.2　软件工程的重要性

随着日益增长的软件需求和软件系统功能的增强，过去一个人开发的历史已不复存在。现在单枪匹马写程序也只是一种娱乐。一般的开发系统都是由一个团队才能完成的，所以必须采用软件工程的方法来设计和开发软件。软件工程中的开发流程是开发出好的软件的前提条件，没有好的流程和管理一定不能开发出好的软件。一个成功的软件不一定是最好的技术，但在背后一定有一个好的流程和管理。所以现代的软件开发，技术不是关键，软件的管理才是主要内容。

把软件设计的整体放在首位，而不去花太多的时间在不一定成功的技术上。如果花太多的时间在技术上，这将对系统的按时完成带来影响。如果要加入新的技术，必须在分析时就预算其所需要的时间，并设置技术风险管理。如果风险太大就应当取消用这项技术，改用其他已成功的技术代替。风险管理是近来才提出的软件管理方法，其对我们的软件项目有着很好的控制作用。

众所周知，软件开发中有太多的不可预知性存在。但这种不可预知是对总体来说的，当软件进行到一定程度时，不可预知的东西就会变成可预知的东西。以往的做法是不去管理，但这样所带来的结果就是项目的失败。如果用软件工程的管理方法来控制这些不可预知的东西，那么软件项目就会一步步随着设计思路走向成功。

软件工程有三种科学管理方法来保证项目的成功，分别如下所述。

1．错误管理，从错误中吸取经验教训

软件开发是一项复杂的、长期的活动。一个典型的软件开发项目会提供很多的机会从错误中吸取经验教训。所以在软件开发中少不了要对错误进行管理。在项目的错误管理中有如下几种做法：

1）列出典型错误

过程方面的典型错误。如过于乐观的项目计划（时间和人员配置不够）、缺乏足够的项目风险管理（对于会出现的风险没有预计）、缺乏整体计划（没有制定长期的目标和实施方案）、在各种压力下放弃原订计划（为赶进度，放弃回归测试）、在模糊的项目前期浪费过多的时间（力求做到完美的需求分析，但这是不可能的）、缺少管理控制（对于各种过程和人员没有管理方法）、缺少质量保证措施（没有对文档和代码进行审查的机制）、鲁莽编码（写出超过预定代码行数的代码）等。

技术方面的典型错误。例如，过高估计了新技术或方法带来的节省量（采用 JAVA 开发不一定比 VC 开发快）、项目中间切换工具（前期采用 VC 开发，后期改变为 JAVA）、缺乏自动的源代码控制手段（没有源代码管理工具和软件）等。

人员方面的典型错误。例如，对有问题的员工管理失控（核心开发人员流失）、挫伤工作积极性、人员素质低、个人英雄主义、项目的后期加入人员、开发人员与用户之间发生摩擦、不现实的预期、缺乏有效的项目支持、缺乏各种角色的齐心协力、政治高于物质、充满想象等。

2）列出自己的最差实践

除了以上的典型错误外，还需要建立自己的最差实践列表，可以避免在以后的项目中犯同样的错误。

3）列出项目中的最差实践

组织机构和其他项目组总结经验，学习错误中得到的经验。和其他组同事交流项目开发的经验，列出潜在的错误，这样就可以尽量避免今后犯同样的错误。

公司在发展的同时，也会积累一些各方面的经验。列出所有的经验，并将其分类系统分析中的经验提供给系统分析，设计人员中的经验提供给管理人员，技术中的经验提供给开发员。这样就会有更多的时间花在新的错误防范上，后面开发出来的系统就会一个比一个好。

2．风险管理，提高了软件开发的成功性

软件项目所面临的不断变换的用户需求、糟糕的计划与评估、欠缺的管理经验、人员问题、伤筋动骨的技术失败、性能欠佳等不胜枚举的风险，使大型项目按时完成的概率几乎为零。

所以项目开发中对风险进行控制管理就大大提高了软件开发的成功性。软件风险管理工作就是在风险成为影响软件项目成功的威胁之前，识别、着手处理并消除风险的源头。

风险管理从最差到最好分为如下 5 种层次：

- ❑ 危机管理——救火模式，就是在风险已经造成麻烦后才着手处理。
- ❑ 失败处理——察觉到了风险并迅速做出反应，但只是在风险发生之后。
- ❑ 风险缓解——事先制定好风险发生后的补救措施，但不做任何防范措施。
- ❑ 着力预防——将风险识别与风险防范作为软件项目的一部分加以规划和执行。
- ❑ 消灭根源——识别和消除可能产生风险的根源。

前面 3 项都是被动进行的风险管理的层次，属于亡羊补牢，为时已晚。所以应该把主要精力放在着力预防风险和消除风险根源层次上。

风险管理由风险评估和风险控制两部分组成。而风险评估又由风险识别、风险分析和风险优先级组成。

- ❑ 风险识别：就是提出一个潜在破坏项目进度的风险列表，类似于错误列表。
- ❑ 风险分析：评估每一个风险出现的可能性及其影响，判定风险的级别。
- ❑ 风险优先级：按风险影响大小排出一个风险优先级，这个风险列表将作为风险控制的基础。

风险控制由风险管理计划，风险化解和风险监控组成。

（1）风险管理计划：制定一个应对每个重要风险的方案，同时确保每一个单独的风险管理计划之间以及与整体项目计划之间相一致。

（2）风险化解：每个重要风险所对应计划的执行。

（3）风险监控：就是对解决风险的过程进行监控，风险监控还可以包括识别新的风险并将其反馈到正在进行的风险管理进程中等方面的工作。

现在以笔者以前做的项目来说明一下是怎样进行风险管理的。

接到项目对项目进行调研工作，在调研中就要注意克服错误列表中的错误。调研完成后，写需求说明书初稿（一般根据情况至少给出两个以上的方案），为用户进行讲解，结合

用户意见再次进行修改。把修改后的说明书和同事进行讨论，再次进行修改。在此期间写出总体设计的初稿（大的框架）。最后再为用户讲解，再次修改少量的功能。用户确定需求满足后就可进行总体设计了。

在生成需求分析的同时，注意列出需求中存在的风险。如需求改变问题、需求定义欠佳等风险。在进行总体设计时，多和用户交流。因为在总体设计中修改需求比在详细设计中修改要容易，比在编码阶段修改就更加容易了。之后生成总体设计说明书。同时在总体设计中也要对一些不确定的因素进行风险监控，列出风险列表。根据总体设计说明书就可以开始详细设计了。

在详细设计中除了要考虑系统设计外还要考虑一些技术风险问题，把很难预见的问题列到风险列表中。注意，从需求分析到详细设计，随着系统开发的进度，以前不确定的各种因素将会慢慢显露出来。同时也会出现新的不确定因素。这样就让笔者必须在整个设计开发过程中进行风险监控、风险识别、风险分析和风险化解工作。

同理，在编码中也同样处理。在开发过程中根据分析不同，把风险按阶段分为需求分析阶段风险、总体设计阶段风险、详细设计阶段风险和编码阶段风险。并交由此阶段的人员进行监控和化解。同时，如果在化解安全区（规定解决问题的时间段中）内无法完成解决，则在讨论会上进行解决。

当然软件开发中所碰到的风险是很多的，不可能完全同时进行风险监控，通常是把风险列表中认为最会发生的风险进行严格的监控起来。随着开发进度，风险是在变化的，所以风险列表可能会增加也可能会减少。只要风险管理好了，软件开发就成功了一大半。

3. 人员管理

不同人员之间经验的不同导致绩效差别是有目共睹的，一些明确激励措施会带来积极的正面影响，所以人员管理在软件项目中也有较重的分量。很明显，人力因素极大地影响着生产效率，同时任何关注提高生产效率的组织首先必须有一套良好的人员激励、团队合作、员工选择及培训的机制。这样才能充分发挥人员的自身能动性，为公司创造更多的价值。

现在的项目管理中，要求开发人员“按章办事”，否则其也只是一部会编程的机器人而已。前面已讲了很多软件工程的重要性，这里不再赘述。现在打个比喻，如果把软件工程比做音乐家，那么项目管理就是音乐指挥家。一个好的音乐家一个人能演奏出动听的音乐，但一群好的音乐家在一起不一定能演奏出好的交响乐，还必须有一位好的指挥家。软件开发也是一样的，有好的程序员只是前提条件，要开发出好的软件，还要有一个好的项目管理方法。

6.2.3　软件工程管理的流程

软件从开发直到报废的整个生命周期被称为软件生存周期。软件工程从时间角度对软件开发和维护的复杂问题进行分解，把软件生命的漫长周期依次划分为若干流程处理阶段，每个阶段有相对独立的任务，然后逐步完成每个阶段的任务。软件生存周期受软件规模、种类、开发方式、开发环境、方法论的影响有两种划分方法，如下所示。

（1）系统分析时期（包括问题定义、可行性研究、需求分析）；软件设计时期（包括总体设计、详细设计、编码和单元测试、综合测试）；软件使用时期与维护时期，各阶段的

关键问题和阶段性成果如表 6.1 所示。

表 6.1　各阶段的关键问题和阶段性成果

阶　段	关 键 问 题	成　果
问题定义	需要解决的问题是什么	规模和目标报告书
可行性研究	有办法解决问题吗	高层逻辑模型 数据流图 成本和效益分析报告
需求分析	软件必须做什么	系统的逻辑模型 数据流图 数据字典（类清单、对象间关系） 算法描述
总体设计	概括性描述如何解决问题	描述可能的解决方法 系统流程图 系统结构（层次图，结构图）
详细设计	怎样具体实现这个系统	编码规格说明
综合测试	是否是符合要求的软件	综合测试方案和结果 测试用例列表 完整性一致的软件配置
维护	持久地满足用户需求的软件	完整准确的维护记录

（2）把软件生命周期划分为 6 个阶段：制定计划、需求分析、软件设计、程序编写、软件测试和运行维护。各阶段的划分和主要任务如表 6.2 所示。

表 6.2　各阶段的阶段性成果

阶　段	主 要 任 务
制定计划	软件开发方与需求方共同讨论，确定软件的开发目标、可行性及制定项目计划
需求分析	在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段是一个很重要的阶段，这一阶段做得好，将为整个软件开发项目的成功打下良好的基础。需求是在整个软件开发过程中不断变化和深入的，因此必须制定需求变更计划来应付这种变化，以保护整个项目的顺利进行
软件设计	主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计，数据库设计等。软件设计一般分为总体设计和详细设计。好的软件设计将为软件程序编写打下良好的基础
程序编写	将软件设计的结果转换成计算机可运行的程序代码。在程序编码中必须要制定统一，符合标准的编写规范。以保证程序的可读性、易维护性，提高程序的运行效率
软件测试	软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分单元测试、组装测试以及系统测试3个阶段进行测试的方法主要有白盒测试和黑盒测试两种。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性
运行维护	软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求。要延续软件的使用寿命，就必须对软件进行维护。软件的维护包括纠错性维护和改进性维护两个方面

6.3　需 求 分 析

俗话说“万事开头难”。需求分析就是项目开始的第一步，只有做好了需求分析才能做出好的、成功的项目。需求分析按照由顶至底、由大到小、由粗到精的过程来进行。需求分析是整个实施过程中至关重要的一步，是否制定出合理的用户需求决定了以后整个系统实施的成败。

6.3.1　什么是需求分析

需求分析是指充分了解用户情况及理解用户的需求，就软件功能与用户达成一致，并与用户一起讨论对系统的具体要求，估计软件风险和项目代价，最终形成开发计划的一个复杂过程。

在这个过程中，用户是处在主导地位的，需求分析工程师和项目经理主要负责整理用户需求，为之后的软件设计打下基础。需求分析阶段结束后，需要制作需求分析说明书、项目可行性报告等文档。

6.3.2　需求分析的任务和过程

简单地说，需求分析的任务就是解决“需要做什么”的问题，就是要全面地理解用户的各项要求，并准确地表达所接受的用户需求。需求分析的整个过程可分为如下 4 个方面。

1．问题识别

是指从系统角度来理解软件，确定对所开发系统的综合要求，并提出这些需求的实现条件，以及需求应该达到的标准。这些需求包括功能需求（做什么）、性能需求（要达到什么指标）、环境需求（如服务器或者用户端的电脑机型，采用的操作系统等）、可靠性需求（保证不发生故障的概率）、安全保密需求（采用加密方法、密码最小长度等）、用户界面需求、资源使用需求（软件运行是所需的内存和 CPU 等）、软件成本消耗与开发进度需求、预先估计以后系统可能达到的目标。

2．分析与综合

逐步细化所有的软件功能，找出系统各元素间的联系，接口特性和设计上的限制，分析其是否满足用户需求，剔除不合理的部分，增加需要的部分。最后，综合成系统的解决方案，给出要开发系统的详细逻辑模型。

3．制定规格说明书

即编制文档，描述需求的文档称为软件需求规格说明书。请注意，需求分析阶段的成果是需求规格说明书，并向下一阶段提交。

4. 评审

对功能的正确性、完整性和清晰性，以及其他需求给予评价。评审通过后才可进行下一阶段的工作，否则重新进行需求分析。

6.3.3　需求分析的方法

需求分析的方法有很多，在这里笔者只给出比较实用和通用的原型化方法，其他的方法，例如结构化方法、动态分析法等在此不讨论。

原型化方法是十分重要的需求分析方法。这里所谓的原型就是指软件的一个早期可运行的版本，其实现了目标系统的某些或全部功能。

原型化方法就是尽可能快地建造一个粗糙的系统，这个系统实现了目标系统的某些或全部功能，但是这个系统可能在可靠性、界面的友好性或其他方面上存在缺陷。建造这样一个系统的目的是为了考察某一方面的可行性，如算法的可行性、技术的可行性或者考察是否满足用户的需求等。

例如：为了考察是否满足用户的要求，可以用某些软件工具快速地建造一个原型系统，这个系统只是一个操作界面，然后听取用户的意见，改进这个原型。以后的目标系统就在这个原型系统的基础上开发出来。

原型主要有如下 3 种类型：

- 探索型，其目的是要弄清楚对目标系统的要求，确定所希望的特性，并探讨多种方案的可行性。
- 实验型，其目的是用于大规模项目开发和实现前，考核制作的计划和方案是否合适，需求规格说明是否可靠。
- 进化型，其目的不在于改进需求规格说明，而是将系统建造得易于变化，在改进原型的过程中，逐步将原型进化成最终系统。

在使用原型化方法时有如下两种不同的策略：

- 废弃策略，其是先建造一个功能简单而且质量要求不高的模型系统，针对这个系统反复进行修改，形成比较好的算法思想，然后据此设计出较完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被废弃不用。前面说的探索型和实验型属于这种策略。
- 追加策略，先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求发展成为最终系统。前面说的进化型属于这种策略。

6.3.4　需求分析的 20 条法则

其实用户与开发人员交流需要好的方法。下面笔者给出 20 条建议，用户和开发人员可以通过评审以下内容并达成共识，以便减少以后的摩擦。

1. 分析人员要使用符合用户语言习惯的表达

需求讨论集中于业务需求和任务，因此要使用术语。用户应将有关术语（例如采价、

印花商品等采购术语）教给分析人员，而用户不一定要懂得计算机行业的术语。

2．分析人员要了解用户的业务及目标

只有分析人员更好地了解用户的业务，才能使产品更好地满足需要。这将有助于开发人员设计出真正满足用户需要并达到期望的优秀软件。为帮助开发和分析人员，用户可以考虑邀请用户观察自己的工作流程。如果是切换新系统，那么开发和分析人员应自己使用一下目前的旧系统，这样有利于相关人员了解目前系统是怎样工作的、其流程情况，以及可供改进之处。

3．分析人员必须编写软件需求报告

分析人员应将从用户那里获得的所有信息进行整理，以区分业务需求及规范、功能需求、质量目标、解决方法和其他信息。通过这些分析，用户就能得到一份“需求分析报告”，此份报告使开发人员和用户之间针对要开发的产品内容达成协议。报告应以一种用户认为易于翻阅和理解的方式组织编写。用户要评审此报告，以确保报告内容准确完整地表达其需求。一份高质量的“需求分析报告”有助于开发人员开发出真正需要的产品。

4．要求得到需求工作结果的解释说明

分析人员可能采用了多种图表作为文字性“需求分析报告”的补充说明，因为工作图表能很清晰地描述出系统行为的某些方面，所以报告中各种图表有着极高的价值。虽然工作图表不太难于理解，但是用户可能对此并不熟悉，因此用户可以要求分析人员解释说明每个图表的作用、符号的意义和需求开发工作的结果，以及怎样检查图表有无错误及不一致等。

5．开发人员要尊重用户的意见

如果用户与开发人员之间不能相互理解，则关于需求的讨论将会有障碍。共同合作能使大家“兼听则明”。参与需求开发过程的用户有权要求开发人员尊重其为项目成功所付出的时间。同样，用户也应对开发人员为项目成功这一共同目标所做出的努力表示尊重。

6．开发人员要对需求及产品实施提出建议和解决方案

通常用户所说的“需求”已经是一种实际可行的实施方案，分析人员应尽力从这些解决方法中了解真正的业务和技术需求，同时还应找出已有系统与当前业务不符之处，以确保产品不会无效或低效；在彻底弄清业务领域内的事情后，分析人员就能提出相当好的改进方法，有经验且有创造力的分析人员还能提出并增加一些用户没有发现的很有价值的系统特性。

7．描述产品使用特性

用户可以要求分析人员在实现功能需求的同时注意软件的易用性，因为这些易用特性或质量属性能使用户更准确、高效地完成任务。

例如，用户有时要求产品要“界面友好”或“健壮”或“高效率”，但这些信息对于开发人员来讲，是主观无实用价值的信息。正确的做法是，分析人员通过询问和调查了解

用户所要的"友好"、"健壮"、"高效"所包含的具体特性，具体分析哪些特性对哪些特性有负面影响，在性能代价和所提出解决方案的预期利益之间做出权衡，以确保做出合理的取舍。如界面友好（菜单和快捷键多）、健壮（长时间使用不出错）、高效率（运算速度快）。

8．允许重用已有的软件组件

需求通常有一定的灵活性，分析人员可能发现已有的某个软件组件与用户描述的需求很相符，在这种情况下，分析人员应提供一些修改需求的选择以便开发人员能够降低新系统的开发成本和节省时间，而不必严格按原有的需求说明进行开发。所以说，如果想在产品中使用一些已有的商业常用组件，而其并不完全适合用户所需的特性，这时一定程度上的需求灵活性就显得极为重要了。

9．要求对变更的代价提供真实可靠的评估

有时，人们面临更好、也更昂贵的方案时，会做出不同的选择。而这时，对需求变更的影响进行评估是十分必要的，可以对业务决策提供帮助。所以，用户有权利要求开发人员通过分析给出一个真实可信的评估，包括影响、成本和得失等。开发人员不能由于不想实施变更而随意夸大评估成本。

10．获得满足用户功能和质量要求的系统

每个人都希望项目成功，但这不仅要求用户要清晰地告知开发人员关于系统"做什么"所需的所有信息，而且还要求开发人员能通过交流了解清楚取舍与限制。一定要明确说明你的假设和潜在的期望，否则，开发人员开发出的产品很可能无法让你满意。

11．用户要给分析人员讲解业务

分析人员要依靠用户来讲解业务概念及术语，但用户不能指望分析人员会成为该领域的专家，而只能让其明白用户自己的问题和目标；不要期望分析人员能将用户业务的细微之处全部描述出来，因为其可能对于用户来说是一些常识，所以根本就不会对分析人员描述。

12．抽出时间清楚地说明并完善需求

用户很忙，但无论如何用户有必要抽出时间参与需求相关会议的讨论，接受分析人员的采访或其他获取需求的活动。有些分析人员可能先明白了用户的观点，而过后发现还需要用户的讲解，这时请耐心对待一些需求和需求的精化工作过程中的反复，因为其是人们交流中很自然的现象，何况这对软件产品的成功极为重要。

13．准确而详细地说明需求

编写一份清晰、准确的需求文档是很困难的。因为处理细节问题不但烦人而且耗时，需要长时间的沟通。因此很容易留下模糊不清的需求。但是在开发过程中，必须解决这种模糊性和不准确性，而用户恰恰是为解决这些问题做出决定的最佳人选，否则，就只好靠开发人员去猜测了。

可以在模糊不清的需求分析中暂时加上"待定"标志，这是一个有效的好方法。用该标志可指明哪些是需要进一步讨论、分析或增加信息的地方，有时也可能因为某个特殊需

求难以解决或没有人愿意处理其而标注上“待定”。用户要尽量将每项需求的内容都阐述清楚，以便分析人员能准确地将这些需求写进“软件需求报告”中去。如果用户一时不能准确表达，通常就要求用原型技术，通过原型开发，用户可以同开发人员一起反复修改，不断完善需求定义。

14．用户必须及时做出决定

分析人员会要求用户做出一些选择和决定，这些决定包括来自多个用户提出的处理方法或在质量特性冲突和信息准确度中选择折衷方案等。有权做出决定的用户必须积极地对待这一切，尽快做处理、做决定，因为开发人员通常只有等用户做出决定才能行动，而这种等待会延误项目的进展。

15．尊重开发人员的需求可行性及成本评估

所有的软件功能都有其成本。用户所希望的某些产品特性可能在技术上行不通，或者实现这个特性需要付出极高的代价，而某些需求试图达到在操作环境中不可能达到的性能，或试图得到一些根本得不到的数据。开发人员会对此做出负面的评价，用户应该尊重开发人员的专业意见。

16．划分各级需求的优先级

绝大多数项目没有足够的时间或资源实现功能性的每个细节。决定哪些特性是必要的，哪些是重要的，是需求开发的主要部分，这只能由用户负责设定需求优先级，因为开发者不可能按照用户的观点决定需求优先级。开发人员只能为用户确定好的优先级提供有关每个需求的花费和风险的信息。

在时间和资源限制下，关于所需特性能否完成或完成多少应尊重开发人员的专业意见。尽管没有人愿意看到自己所希望的需求在项目中未被实现，但毕竟要面对现实。业务决策有时不得不依据优先级来缩小项目范围或延长项目开发工期，或增加资源，或在质量上寻找折衷。

17．用户要慎重认真地评审需求文档和原型

因为用户评审需求文档，是给分析人员带来反馈信息的一个机会。如果用户认为编写的“需求分析报告”不够准确，就有必要尽早告知分析人员并为改进提供建议。

更好的办法是先为产品开发一个原型。这样用户就能提供更有价值的反馈信息给开发人员，使其更好地理解用户的需求；原型并非是一个实际应用产品，但开发人员能将其转化、扩充成功能齐全的系统。

18．如果用户的需求变更必须要立即联系

不断的需求变更，会给在预定计划内完成的产品质量带来严重的不利影响。变更是不可避免的，但在开发周期中，变更越在晚期出现，其影响越大；变更不仅会导致代价极高的返工，而且工期将被延误，特别是在大体结构已完成后又需要增加新特性时。所以，一旦用户发现需要变更需求时，请立即通知分析人员。

19. 遵照开发小组处理需求变更的过程

为将变更带来的负面影响减少到最低限度，所有参与者必须遵照项目变更控制过程。这要求不放弃所有提出的变更，对每项要求的变更进行认真的分析、综合考虑，最后做出合适的变更决策，以确定应将哪些变更引入项目中。

20. 尊重开发人员采用的需求分析过程

软件开发中最具挑战性的莫过于收集需求并确定其正确性，分析人员采用的方法有其合理性。也许用户认为收集需求的过程不太划算，但请相信花在需求开发上的时间是非常有价值的；如果用户理解并支持分析人员为收集、编写需求文档和确保其质量所采用的技术，那么整个过程将会更为顺利。

6.3.5　深入获得用户的需求

除了前面说的方法外，还要注意大多数的用户一般对于软件设计都不是很了解，只能提供语言描述。为此，必须做到如下几点：

- 学会使用用户的语言来描绘软件产品，这样用户才能知道软件项目设计者所设计的软件是否符合其需求。
- 学会理解用户的身份。因为每一个人所处的工作位置不同，其提出的软件需求也是不同的。要抓住主要的使用者所提的需求。
- 了解用户的价值观，注意用户对软件界面上的需求。
- 理解用户需求背后的深层次心理需求，例如，软件除了能够帮助用户解决现实中的问题外，还可以提供什么方便。
- 像用户一样体验，贴近用户的使用习惯。例如，有的用户喜欢表浏览的模式，有的用户喜欢对话框模式。

6.3.6　可行性分析

在了解和分析用户的需求之后，就需要预测整个项目的可行性。因为并不是所有的问题都有简单明显的解决办法。如果问题没有可行的解决办法，那么花费在这项开发工程上的任何时间、资源、人力和成本都是无谓的浪费。可行性分析是要决定“做还是不做”。需求分析是要决定“做什么，不做什么”。

可行性分析的目的就是用最小的代价在尽可能短的时间内确定问题是否能够解决。但有一点笔者要提醒读者。

注意：可行性分析的目的不是解决问题，而是确定问题是否值得去求解。

要达到可行性分析的目的，不能靠主观猜想而只能依靠客观分析。必须分析几种主要能解决问题的方法的利和弊，从而判断原预定的系统目标和规模是否现实，软件系统在完成后所能带来的效益是否大到值得投资开发这个软件系统的程度。

可行性分析实质上是进行一次压缩和简化了的系统分析和设计的过程，是较高层次上

以抽象的方式进行的系统分析和设计的过程。

一般可以从如下 3 个方面来分析项目的可行性：

- 技术可行性，使用现在的软件技术能否实现用户需要的软件系统。
- 经济可行性，这个软件系统的经济效益能超过其开发和设计成本吗？
- 操作可行性，软件系统的操作方式在这个用户组织内行得通吗？

除了以上要求外，分析人员还要为每一个可行性的解法制定一个粗略的实现进度。如果都是可行的，那么分析人员应该推荐一个较好的解决方案，并且为工程制作一个初步的项目计划。

6.3.7　成本效益分析

一般来说，投资开发软件系统的目的是为了再得到更大的经济效益。经济效益通常表现为减少运行成本或者增加收入两个方面。

但投资开发软件系统也是存在一定风险的，有可能软件系统的开发成本可能比预计的高，而效益则比预计的低。那么在什么情况下投资开发软件系统更划算呢？这就是本节要讲解的成本效益分析。

成本效益分析的目的是从经济角度分析开发一个特定的新系统是否划算，从而帮助使用软件系统的组织负责人正确地做出是否投资开发该项目的决定。

1．成本估算

软件开发的成本主要表现为人力消耗（开发人员的工资、工作时间、消耗的物资）和其他开发费用（开发环境的硬件和软件投入）。下面简单介绍 3 种估算技术。

1）代码行技术

代码行技术是比较简单的定量估算方法，其是把开发软件的每一个功能的成本和实现这个功能需要用的源代码行数联系起来。通过经验和历史数据估计实现一个功能需要的源程序行数。

一旦估计出源代码行数后，用每行代码的平均成本乘以行数就可以确定整个软件系统的成本。每行代码的平均成本主要取决于软件的复杂度和开发人员的工资水平。

2）任务分解技术

这种方法是把软件开发工程分解为若干个相对独立的任务。再分别估计每个单独开发的任务成本，最后累加起来得到软件开发工程的总成本。估计每个任务的成本时，通常先估计完成该任务需要有的人力成本，再乘以每人每月的平均工资而得出每个任务的单个成本。典型环境下的各开发阶段需要使用的人力比例如表 6.3 所示。

3）自动估计成本技术

采用自动估计成本的软件工作可以减轻人力，使得估计的结果更客观。但是采用这种技术必须有长期搜集的大量历史数据为基础，并且需要有良好的数据库系统作为支持。

2．效益预计

除了估计开发成本外，还要预计软件系统带来的经济效益。这个经济效益等于因使用软件系统而增加的收入加上使用软件系统可以节省的运行费用。而且还要合理地估计软件

表 6.3　典型环境下各阶段需要使用的人力比例

阶 段 任 务	人 力 比 例
需求分析	10%
可行性分析	5%
设计	25%
代码编写	20%
测试与维护	40%
总计	100%

系统的使用寿命。一般为了保险一些，都设定周期为 5 年。

预计软件系统带来的经济效益可以从以下 3 个方面计算。

1）货币的时间价值

通常用当年银行利率来表示货币的时间价值。

2）投资回收期

投资回收期是衡量一项开发项目的价值的主要标准。其就是累计的经济效益等于最初投资所需要的时间。显然，投资回收期越短就越快获得利润，该项目就越值得投资。

3）纯收入

纯收入是衡量项目价值的另一个主要标准。其是指整个软件生命周期同系统的累计经济效益与投资之差。

6.3.8　确定开发环境

这一步是开发软件系统时必须要做出的一个重要决定。使用什么开发环境，直接决定了项目开发的时间和人力投入。开发环境的选择有如下几个标准。

1．系统用户的要求

如果开发出来的软件系统由用户自己负责维护，用户通常要求用其熟悉的语言书写代码程序。

2．可以使用的编译程序

运行操作系统的环境中可以提供的编译程序往往限制了可以选用的语言的范围。

3．使用主流的软件开发工具编写代码

如果某种编程语言由主流的软件开发工具所支持，那么其代码的编写和调试都会变得比较容易。

4．程序员的知识

虽然对于有经验的程序员来说，学习一种语言并不困难，但是要完全掌握一种新语言却需要非常长的实践时间。应该选择一种程序员所熟悉的编程语言来开发和设计软件系统。

5．软件可移植性要求

如果软件系统将在多台不同的电脑上运行，或者使用寿命很长时，应该选择一种标准化程度高、可移填性好的语言来开发。

6．软件系统的应用领域

其实，所有的编程语言实际上并不是对所有应用领域都适用的。例如：FORTRAN 语言特别适合于工程和科学计算；C++适用于游戏、系统和实时应用领域；Java 适应于网络和跨平台。

使用什么程序语言的开发环境，决定了实现这个软件系统能够使设计完成编码时困难最少，可以减少需要的程序测试量，并且可以得出更容易阅读和更容易维护的程序。而且由于软件系统的绝大部分成本用在测试和维护阶段，所以选择容易测试和维护的代码编写语言是极其重要的。

6.4　项目计划安排

在可行性分析之后，项目计划安装将贯穿于整个软件工程的每一个环节。其是软件工程非常重要的一部分。

6.4.1　项目开发计划的重要性

由于软件开发是一种智力创作活动，很难像传统工业那样通过执行严格的操作规范和时间定量来保证软件产品实现的时间计划。还有其他因素都会影响项目开发进度，例如需求变更、人员流动、各种风险等因素。

所以必须制定和提供一份合理的进程表（项目开发计划），来保证项目的顺利进行。项目开发计划可以让所有开发人员任务明确、步调一致，最终共同准时地完成整个项目。软件的项目计划重在“准确”而非“快速”。

6.4.2　如何制定项目开发计划

制定项目计划，如同预言未来一样困难。如果允许在整个项目结束后再写计划，那就轻松多了，并且可以百分百地准确。

对项目的时间限制有两类。第一类，项目应该完成的日期应写在合同中，如果延期了，则开发方要作出相应的赔偿。第二类是开发自己的软件产品，虽然只确定了该产品大致的发行日期并允许有延误，但如果拖延太久则会失去商机造成损失。所以在制定项目计划时要做到如下两点。

1．知己知彼

只有“知己知彼”才能做出合理的项目计划。这里“知彼”是指要了解项目的规模、

难度与时间限制。才可以确定应该投入多少人力、物力去做这个项目。“知己”是指要了解有多少可用资源，如可调用的程序员有几个？他们的水平如何？软硬件设施如何？在可行性分析阶段就要考虑这个问题。但不幸的是，人们在深陷项目泥足之前，总难以准确地估计项目的规模与难度。这个经验起到了最重要的作用。

项目的资源分为 3 类：“人”、“可复用的软构件”和“软硬件环境”，如图 6.1 所示。

其中，人是最有价值的资源。项目计划的制定者要确定开发人员的名单，要根据其专长进行分工。可复用的软构件是次有价值的资源。软构件并非一定要用自己的，可以向专业的软件供应商购买。软硬件环境虽然不是最重要的资源，却是必需的资源。原则上软硬件环境只要符合项目的开发要求即可。有些项目可能要用到特殊的设备，则需要事先做好准备，以免需要用时找不到而耽误了时间进度。

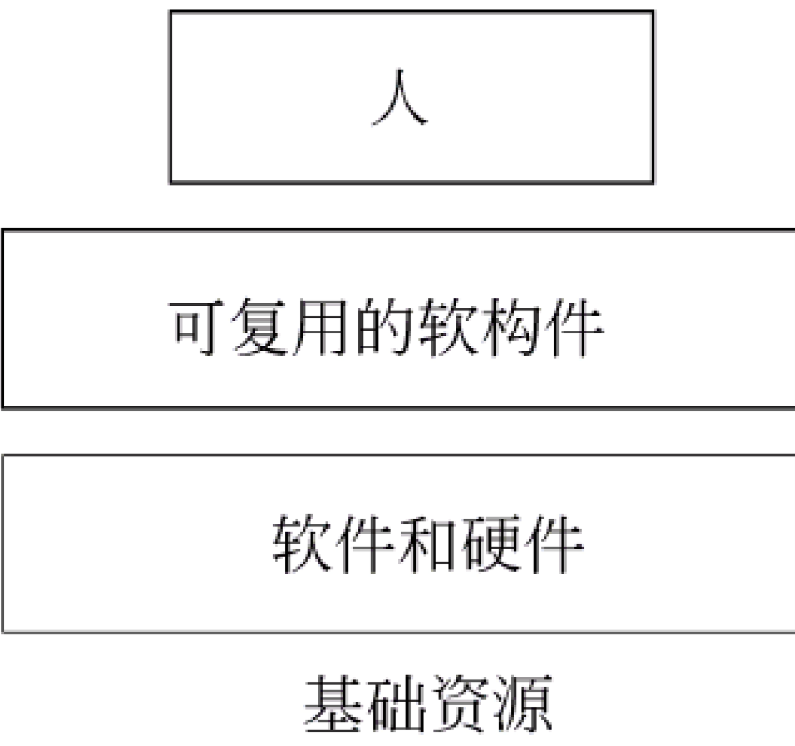


图 6.1 项目资源的组成

2．合理的进度安排

一般开发项目时，实际进度落后于进度表乃是家常便饭，不必大惊小怪。例如以下一些事情经常会导致项目被延误。

- 上级领导主管臆断，制定了不现实的期限。项目经理与程序员们被迫按照不合理的进度表开展工作。
- 客户的需求发生了变化，但没有对进度表做出相应的修改。
- 低估了项目的规模与难度，导致投入的人力和物力不足。
- 并未预见到存在难以克服的技术障碍。
- 并未预见到开发人员会发生问题，如生病、辞职等。
- 开发人员之间不能很好的交流、协作，导致各阶段任务难以如期完成。

所以写进程表笔者提出以下一些有益的建议：

- 制定进度表的人最好就是项目负责人，他应该是最了解项目和开发人员的。进度表要经过开发小组的讨论，在得到大多数人的支持后才能实施，避免出现一厢情愿的局面。
- 进度安排并不见得一定要符合逻辑顺序。应尽可能地先做技术难度高的事，后做难度低的事。也就是“吃苦在前，享乐在后”。
- 开发一个大的软件项目，应该将进度表分为若干个里程碑。一个里程碑之内的多个任务可以同步进行。作为程序员常常极容易沉迷于新技术或者算法的优化，而忘记了时间进度。所以里程碑就像是阶段性的目标，使程序员不迷失开发的方向。
- 进度表中必须留有缓冲时间，并将缓冲时间用到不确定的事情上。因为人们对即将要做的事情知之甚少，所以要留一些时间以防不测。Microsoft 公司的一些开发小组甚至制定了“50% 缓冲规则”。对许多项目经理而言，容忍进度表中存在缓冲时间，不啻为观念上的一个飞跃。
- 如果发现项目应交付的期限非常不合理，就要跟领导或跟客户据理力争，请求放宽期限、调整进度。当客户的需求发生变化时，就要对进度表做出相应的修正。不要觉得修改进度表很困难很麻烦，不修改才会产生真正的麻烦。

6.5　总 体 设 计

经过需求分析和项目计划阶段后，软件系统必须“做什么”已经比较清楚了。现在就是决定“怎样做”的时候了。

6.5.1　总体设计的概念和目的

总体设计的基本目的就是概括地说“软件系统应该如何实现”这个问题，因此，总体设计又被称为概要设计或者初步设计。

6.5.2　总体设计的过程

总体设计的过程是寻找实现目标软件系统的各种不同的方案，需求分析的结果是各种可能方案的基础。总体设计的过程通常有如下几步：

（1）设想供选择的方案。

（2）选取合理的方案。

（3）推荐最佳方案。

（4）功能分解。

（5）设计软件结构。

（6）数据库设计（只对于需要使用数据库的应用领域）。

（7）制定测试计划。

（8）制作系统说明、用户手册、数据库设计结果等文档。

（9）审查和复审前面的总体设计内容。

6.6　详细设计的工具

描述程序处理过程的工具称为详细设计的工具，其可以分为图形、表格和语言 3 种。但无论是哪类工具，对其基本要求都是能提供对设计无歧义的描述。也就是应该能指明控制流程、处理功能、数据组织，以及其他方面的细节，从而在编码阶段能把对设计的描述直接翻译成程序代码。下面将详细介绍程序流程图的知识。

程序流程图又称为程序框图，是历史最悠久，使用最广泛的描述软件设计的方法。但是其也是用得最混乱的一种方法。如图 6.2 所示，列出了一些程序流程图中常用的图形符号。

从上世纪开始，程序流程图一直是软件设计的主要工具之一。其主要优点是对控制流程的描绘很直观，便于初学者掌握。由于程序流程图历史悠久，为人们所熟悉，尽管其有各种缺点，许多人已经建议停止使用，但至今仍在广泛使用着。

技巧：多使用流程图能够将程序交互过程描述得清楚明了。

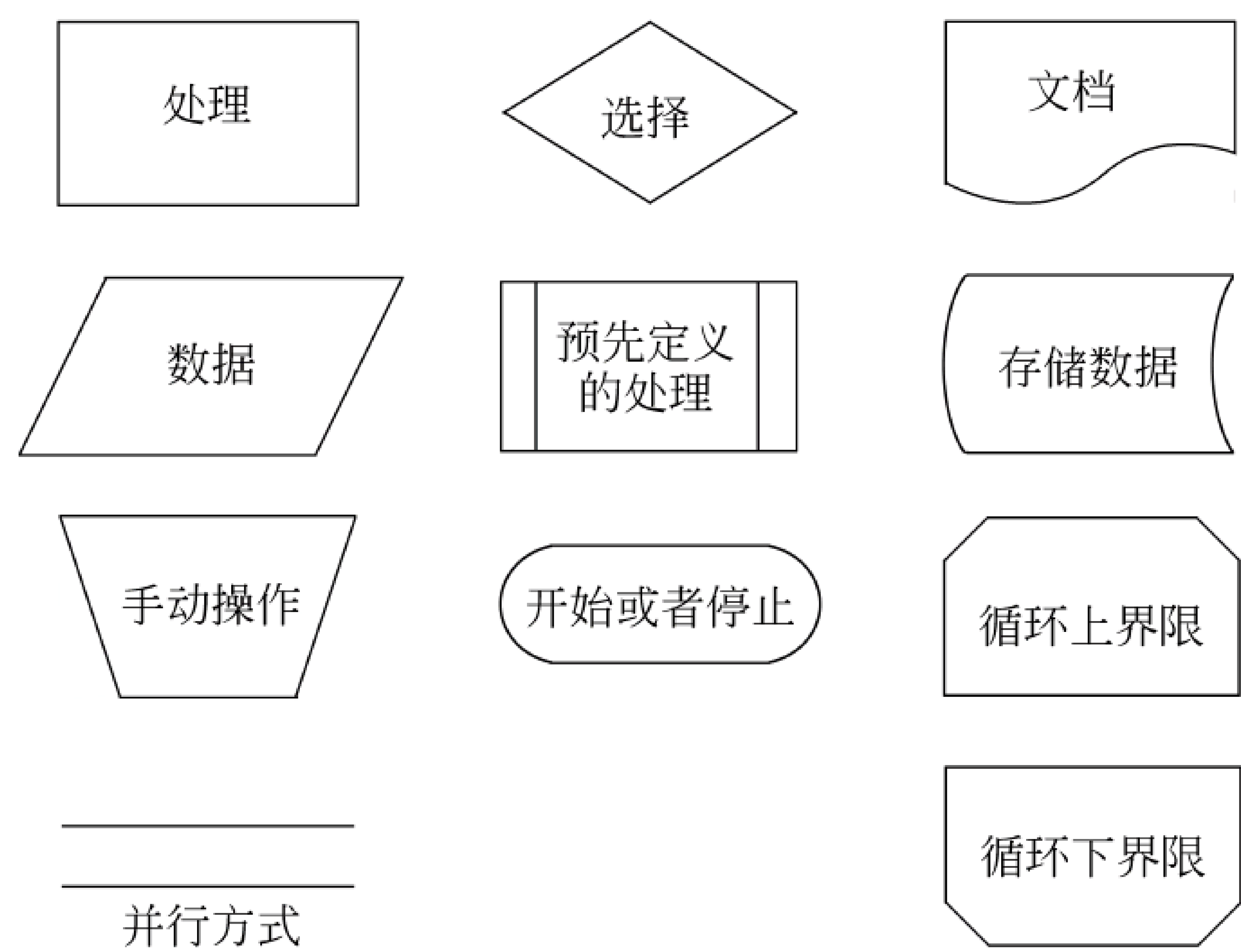


图 6.2　程序流程图中常用的图形符号

程序流程图的缺点如下：

- 程序流程图本质上不是逐步求精的好工具，其让程序员过早地考虑程序的控制流程，忽略了考虑程序的全局结构。
- 程序流程图中用箭头代表控制流，因此程序员可以不受任何约束，可以完全不顾程序结构，随意转移控制。
- 程序流程图不易表示数据结构。

6.7　软件测试

由于在系统开发的漫长过程之中，面对着极其错综复杂的问题，人的主观认识不可能完全符合客观现实。与工程密切相关的各类人员之间的通信和配合也不可能完美无缺，因此，在软件开发的各个阶段都不可避免地会产生差错。所以这时就需要引入软件测试过程，这样才能保证软件的质量。

软件测试是对软件规格说明、设计和代码编写的最后的复审。

6.7.1　软件测试的目标

什么是软件测试？其目标是什么？笔者在这里给出一些关于软件测试的规则，这些规则也可以看作是测试的目标或者定义。

（1）测试是为了发现程序中的错误而执行程序的过程。

（2）好的测试方案是极可能发现迄今为止尚未发现的错误。

（3）成功的测试是发现至今为止尚未发现的错误的测试。

从这些规则中可以看出，软件测试的正确定义是“为了发现程序中的错误而执行程序的过程”。这和通常想象的“测试是为了表明程序是正确的”、“成功的测试是没有发现错误的测试”等是完全相反的。

正确地认识到测试的目标是很重要的，因为测试的目标决定了测试方案。如果为了表示程序的正确性而进行测试，那么就会设计一些不易暴露错误的测试方案；相反，如果测

试只是为了发现程序中的错误，就会力求设计出最能暴露错误的测试方案。

由于测试的目标是为了暴露程序中的错误，所以一般由程序的编写者自己进行测试是不恰当的。因此，通常的做法是由专职的测试人员或者非代码编写人员组成测试小组来完成测试工作。

此外，应该认识到测试决不能证明程序是正确的。即使经过最严格的测试之后，仍然可能还会发现没有被发现的错误潜藏在程序之中。测试只能查找出程序中的错误，而不能证明程序中没有错误。

6.7.2　黑盒与白盒测试

读到这里读者肯定会有疑问，应该怎么对程序进行测试呢？其实测试任何产品都会用到如下两种方法：

- 如果已经知道了产品应该具有的功能，可以通过测试来检验每个功能是否都能正常使用。
- 如果知道产品的内部工作过程，可以通过测试来检验产品的内部动作，是否按照规格说明书的规定正常进行。

在软件测试中，第一种方法被称为黑盒测试；第二种方法被称为白盒测试。

顾名思义，黑盒测试法是把整个程序看成一个黑盒子，完全不考虑程序的内部结构和处理过程。也就是说，黑盒测试是在程序的接口处进行测试，其只检查程序功能是否按照规格说明书的规定正常实现和使用；程序是否能适当地接收输入数据并产生正确的输出信息，保持外部信息的完整性。黑盒测试又被称为功能测试。

与黑盒测试相反，白盒测试法的前提是可以把程序看作是装在一个透明的白盒子里，也就是完全了解程序的结构和处理过程。这种方法按照程序内部的逻辑测试程序，检验程序中的每条通路是否都按预定要求正确工作。白盒测试又被称为结构测试。

在使用黑盒测试时，为了保证测试能发现程序中的所有错误，不仅应该使用有效的输入数据，还必须使用一切可能的输入数据。实践表明，用无效的输入数据进行测试常常比用有效的输入数据进行测试，能发现更多的错误。

而在使用白盒测试时，应该尽可能地把程序中每条可能的通路至少都执行一次。

但由于输入数据的选择性很多，不可能进行真实的穷尽测试。所以软件测试不可能发现程序中的所有错误。即使通过测试也不能证明程序是正确的，只能通过测试保证软件的可靠性。

在设计测试用例时，应力争用尽可能少的测试来尽可以多地发现错误。

6.7.3　软件测试的步骤

一开始就要一个大系统作为单独的实体来测试是不现实的，除非是测试一个很小的程序。所以与开发过程一样，测试的过程也必须分步骤进行。每一个步骤都在前一个步骤上延续。一般大型的软件系统通常由若干个子系统组成，每个子系统又由许多个模块构成。因此一般测试由如下几个步骤组成。

1. 模块测试

在设计得好的软件系统中，每个模块完成一个清晰的子功能，而这些子功能和同层次其他模块的功能之间没有联系。因此，有可能把每个模块作为一个单独的实体来测试，这样比较容易设计检验模块正确性的测试方案。模块测试的目的是保证每个模块作为一个单元能正确运行，所以模块测试又被称为单元测试。

2. 子系统测试

子系统测试是把经过前面单元测试的模块放在一起形成一个子系统来测试。模块相互间的通信和接口是这个测试过程中的主要问题。

3. 系统测试

系统测试是把经过测试的子系统合成为一个完整的系统来测试。在这个测试步骤中发现的往往是软件设计中的错误，也可能发现需求说明中的错误。

4. 验收测试

验收测试是把软件系统作为单一的实体进行测试，测试内容与系统测试基本类似，但其是在用户的参与下进行的，而且主要使用实际数据进行测试。

5. 运行测试

在验收测试后，往往并不会立即投入生产性运行，还要经过一个试用运行时间的考验。这样做的目的可以保证如下几点：

- ❑ 用户能有一段熟悉软件系统的时间。
- ❑ 可以验证用户使用说明之类的文档。
- ❑ 能够对软件系统进行全负荷的测试，用测试结果说明性能指标。

6.7.4　设计测试方案

设计测试方案是整个测试阶段的关键技术问题。所谓测试方案包括预定要测试的功能，应该输入的测试数据和预期的结构。其中，最难的问题是设计测试用例，即输入数据。

不同的测试数据发现程序错误的能力差别很大，为了提高测试效率降低测试成本，应该选用高效的测试数据。因为不可能进行穷尽的测试，选用少量“最有效的”测试数据，做到尽量完备测试就更重要了。

技巧： 软件测试描述的过程越细致，测试人员测试起来才会越明确和有效。

设计测试方案的目标是，确定一组最可以发现某个错误或某类错误的测试数据，并联合多种设计测试数据的技术。

通常的做法是：用黑盒法设计基本的测试方案，再用白盒法来补充方案。

6.8 软 件 维 护

软件维护是软件开发生命周期的最后一个阶段，因为其是处于软件系统投入运行以后的时期中，所以并不同于软件系统开发的过程。本节将主要介绍软件维护的概念及软件项目的可维护性。

6.8.1 软件维护的概念

所谓软件维护就是在软件已经交付用户使用后，为了改正出现的错误或者满足新的需要而修改软件的过程。维护具体可以分为 4 种。

- 因为软件测试不可能暴露整个软件系统所隐藏的错误，所以必然会有软件的维护过程。在任何一个程序的使用期间，用户都会发现程序错误，用户会把其遇到的问题提交给维护人员。这个过程被称为改正性维护。
- 由于电脑科技发展迅速，经常会有新推出的操作系统或者升级包。所以就必须进行软件的适应性维护。
- 在软件系统的使用过程中，用户一般会提出一些改进意见。为了满足这些要求，就需要进行完善性维护。这项维护通常占用软件维护的时间最长。
- 为了改进未来的可维护性和可靠性，给以后的改进奠定更好的基础而修改软件时，就会有预防性维护过程的产生。

6.8.2 软件项目的可维护性

软件可维护性可以定性为：维护人员理解、改正、改动和改进这个软件的难易程度。提高软件的可维护性是支配软件工程的所有步骤的关键目标。

影响软件可维护性的因素主要有如下 3 个方面。

1．可理解性

软件可理解性表现为用户理解软件的结构、接口、功能和内部过程的难易程度。模块化、详细的设计文档、结构化设计、源代码内部的文档和良好的高级程序设计语言等，都对改进软件的可理解性有重要帮助。

2．可测试性

测试的难易程度主要取决于软件容易理解的程度。良好的文档对测试是至关重要的。此外，软件结构、可用的测试工具和调试工具，以及以前设计的测试过程也都很重要。维护人员应该能够得到在开发阶段使用过的测试方案，进行回归测试。在软件的设计阶段也应该尽力把软件设计成为容易测试和维护的。

3．可修改性

软件容易修改的程度与耦合、内聚、局部化、控制域及作用域等因素相关，这些都影响软件的可修改性。

6.9　总　　结

在这一章中，主要介绍了软件开发的整个生命周期，以及各个阶段不同的分析和设计方法。主要应掌握如下内容：

- 项目管理与软件工程的优点及应用。
- 需求分析的方法及法则。
- 如何安排项目计划。
- 总体设计与详细设计的过程。
- 软件测试和维护的重要性及其方法。

最后，通过本章的学习，希望各位读者能够建立起基本的软件项目管理知识。并且在后面的实际项目开发示例中，结合这些知识来理解游戏项目的开发是如何进行的。

从第 7 章开始，笔者将带领大家开始进行真实的项目开发和设计，并把五子棋游戏开发的整个项目分解为几个部分进行讲解。真心地希望各位读者都能够掌握游戏项目的完整开发流程。

第 2 篇　五子棋游戏案例分讲

通过前面内容的学习，各位读者都应该掌握了使用 Visual C++开发一般应用程序的方法，同时对于 Visual C++中声音播放、图像显示都有了一定的了解。但这些只是简单的程序设计，并不涉及真实的项目开发。所以在第 6 章，笔者又把项目管理的相关内容进行了简单介绍。本篇的主要内容就是为了弥补这些不足，对游戏项目开发的各个步骤和知识点进行详细讲解。

本篇是整本书的重点和难点，主要分为 4 章，其中第 7 章是游戏项目管理，主要讲解如何制作五子棋游戏的各种文档。第 8 章是五子棋游戏界面与通信开发详解，内容包括游戏界面的设计与网络通信协议的制作。第 9 章是讲解五子棋游戏的核心算法的设计与实现，这里主要涉及如何实现游戏规则的内容。第 10 章是五子棋游戏的整合测试，主要给出了各种测试用例、文档表格及测试方法。

希望通过本篇的学习，各位读者可以真实有效地掌握好游戏项目的开发、设计与实现的完整流程，并且可以自己动手开发游戏项目。

第 7 章　五子棋游戏项目开发的前期工作

本章主要通过某公司项目开发的实例，来介绍如何进行游戏项目的需求分析、可行性分析、项目计划安排、游戏的概要设计及游戏操作界面设计文档的编写方法。

本章主要涉及的内容如下：

- 游戏项目的需求分析：通过项目实例来了解什么是需求分析及如何进行需求分析。
- 游戏项目的可行性分析：让读者知道什么是可行性分析并且掌握如何进行可行性分析。
- 游戏项目的计划安排：在真实的项目开发中，合理的计划安排是怎样的。
- 游戏项目的概要设计：让读者认识概要设计的文档应该如何制作。
- 游戏项目的操作界面设计：通过实例文档让读者知道如何制作操作界面的设计文档。

7.1　五子棋游戏的用户需求描述

通过与某公司用户的沟通，笔者得到五子棋游戏开发的资料如下所述。

1．五子棋的背景

五子棋是我国古代传统的黑白棋种之一，大约在南北朝时期随围棋一起先后传入朝鲜、日本等地。五子棋在日本又叫“连珠棋”。通过一系列的规则变化使连珠五子棋这个简单的游戏复杂化、规范化，而最终成为今天的职业连珠五子棋，同时也成为一种国际比赛棋。

五子棋不仅能增强思维能力，提高智力，而且富含哲理，有助于修身养性。五子棋既有现代休闲的明显特征“短、平、快”，又有古典哲学的高深学问“阴阳易理”；其既有简单易学的特性，为人民群众所喜闻乐见，又有深奥的技巧和高水平的国际性比赛；五子棋文化源远流长，具有东方的神秘和西方的直观；既有“场”的概念，亦有“点”的连接。是中西文化的交流点，是古今哲理的结晶。

2．五子棋游戏的棋盘

五子棋游戏的棋盘同围棋相同，如图 7.1 所示。棋盘正中一点为“天元”。棋盘两端的横线称端线。棋盘左右最外边的两条纵线称边线。从两条端线和两条边线向正中发展而纵横交叉在第 4 条线形成的 4 个点称为“星”。天元和星应在棋盘上用直径约为 0.5 厘米的实心小圆点来标出。

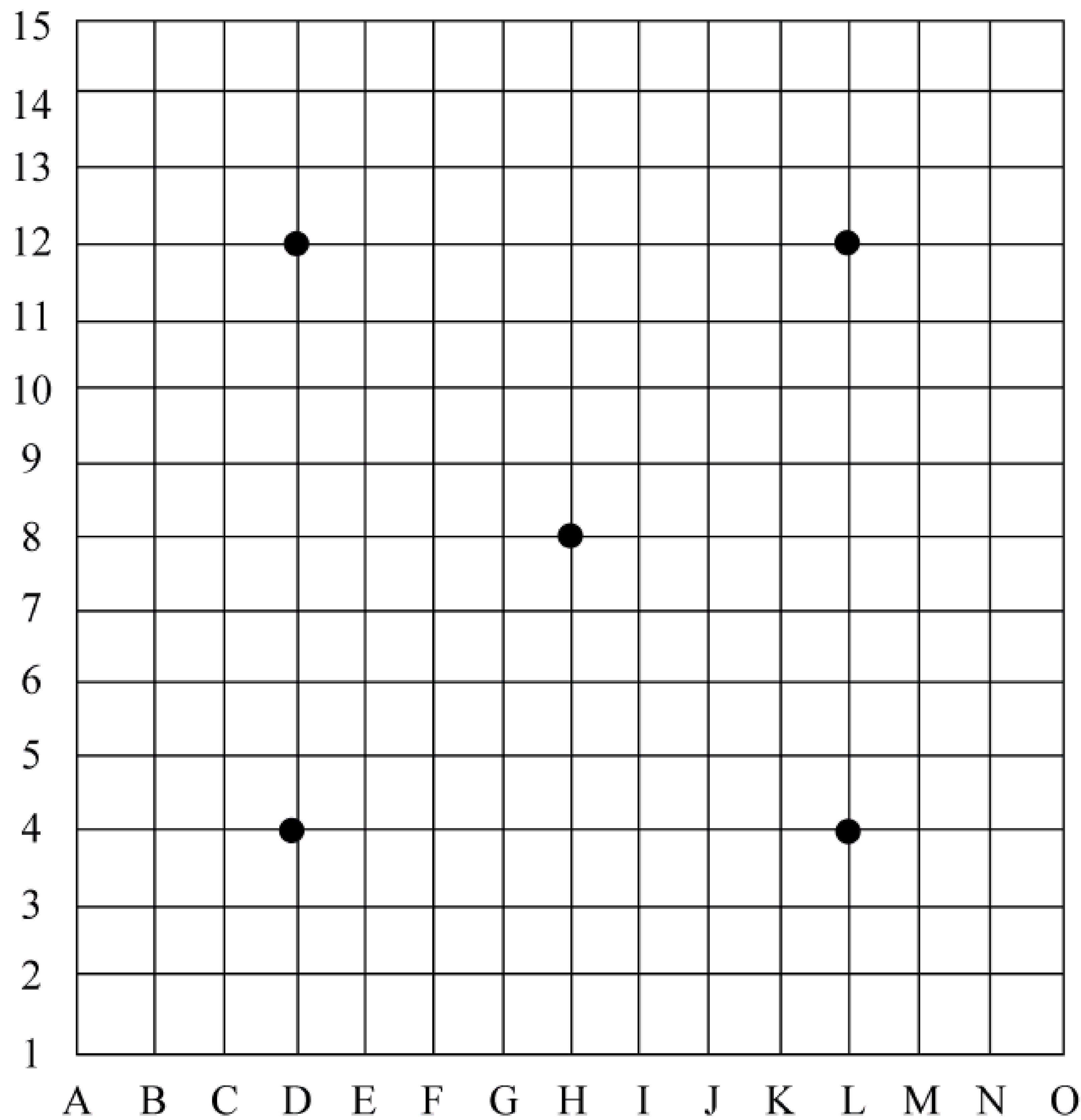


图 7.1　五子棋游戏棋盘

以持黑方为准，棋盘上的纵轴线从左到右用英文字母 A～O 标记。横行线从近到远用阿拉伯数字 1～15 标记。纵横轴上的横纵线交叉点分别用横纵线标记的名称合写成。如“天元”是 H8，四个“星”分别为 D4、D12、L12、L4 等。

3．五子棋游戏的基本规则

黑白双方依次落子，任一方先在棋盘上形成横向、竖向、斜向的连续的相同颜色的五个（含五个以上）棋子的一方为胜。

4．游戏中的其他禁手规则

鉴于无禁手规则黑棋必胜，人们不断采用一些方法限制黑棋先行的优势，以平衡黑白双方的形式。于是针对黑棋的各种禁手逐渐形成。

禁手最简单地说就是一手棋形成长连（连成五个以上连续相同的棋子），或两个以上的活三、或者两个以上的连四，并且这些连四、活三和长连都要包括这一手棋。并且规定，当禁手与连五同时出现时为黑方取胜禁手不成立，禁手只是针对黑棋而言的，白棋没有任何禁手。例如，黑棋长连是禁手，白棋长连算赢棋。

5．用户的其他要求

能够进行网络通信，即一台电脑作为服务器，一台电脑作为客户机。服务器的默认为执黑子。同一时间只由一方落子，另一方等待。当任意一方有连五时，电脑自动通知另一方胜利并提示。

双人网络对弈模式下，在玩家落子后，可以选择和棋。和棋的过程为：首先由玩家向对方玩家发送和棋请求，然后由对方玩家决定是否允许当前玩家和棋，在当前玩家得到对方玩家的响应消息（允许或者拒绝）之后，才进行和棋与否的操作。要注意在发送请求后游戏必须等待对方回应，这里是不允许双方玩家落子的，即游戏处于暂停状态。

7.2　五子棋游戏的需求说明书

通过与用户的沟通和对需求描述的分析，下面就可以开始制作五子棋游戏项目的需求说明书。需求说明书的主要内容是与用户确定五子棋游戏应该具有的功能。

1．引言

某公司为了扩大公司的游戏业务经营范围，需要开发一款支持 Internet 网络的、休闲类对战五子棋游戏。特制定本说明书来用于描述某公司五子棋游戏项目开发的功能性需求。

1.1　编写目的

使用技术性语言，对某公司的五子棋游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。
- 本游戏项目是某公司整个网络游戏系统中的一部分。

2．文档范围

包含某公司五子棋游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司五子棋游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

无。

5．游戏具有的功能

5.1　能够实现五子棋游戏中的全部规则

- 能够对连五的胜负方进行判断，并给出提示，连五如图 7.2 所示。
- 能对黑方禁手进行判断（三三禁手和四四禁手），如图 7.3 所示，并能够在落子后进行直接判负的提示。

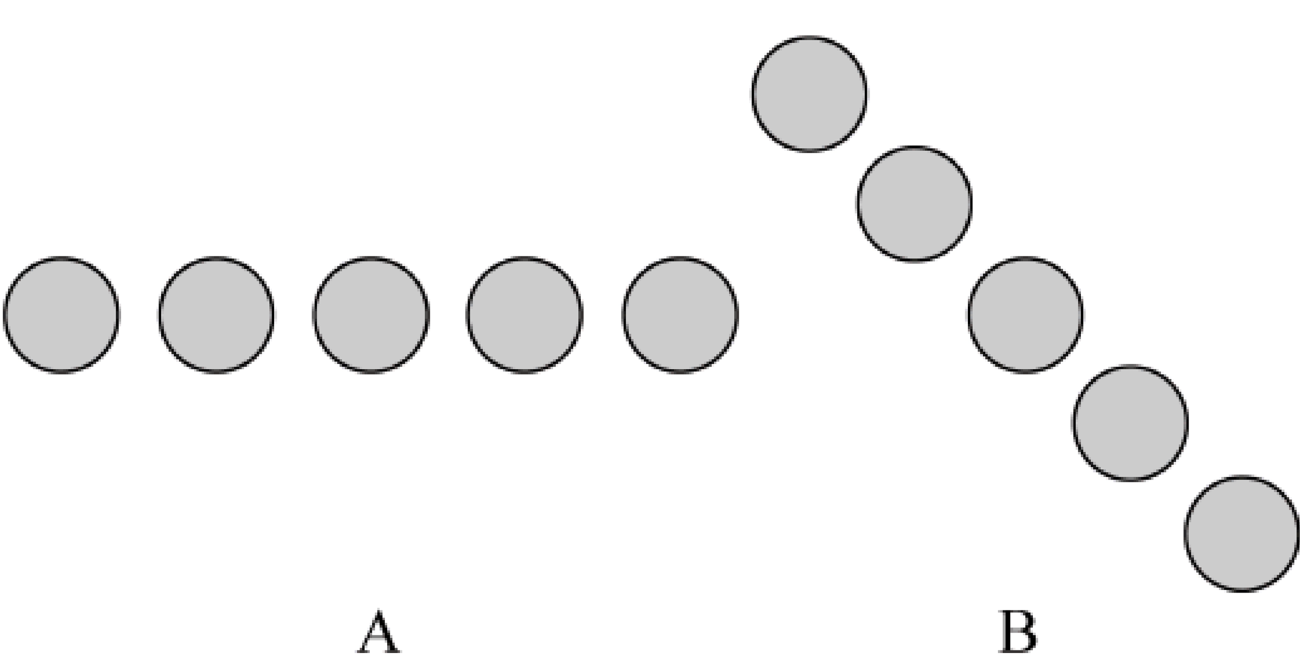


图 7.2　连五

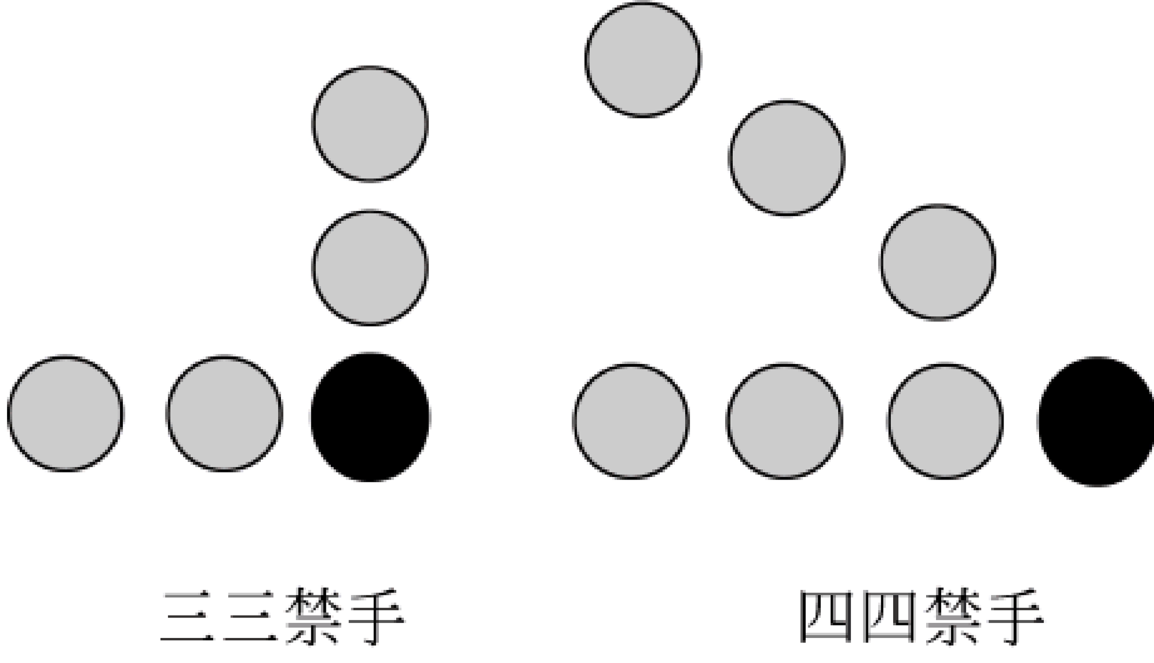


图 7.3　三三禁手和四四禁手

5.2　能够支持设置网络端口

玩家可以自行设定连接的服务器 IP 地址。通信端口默认使用 5005。

5.3　能够支持网络对战

- 支持两个玩家进行联机网络对弈模式。同一时间只能由一方进行游戏，另一方等

待，与真实的五子棋游戏相同。

- 支持和棋操作。和棋操作的过程为：首先由玩家向对方玩家发送和棋请求，然后由对方玩家决定是否允许和棋。如果对方玩家允许和棋，就直接给出提示，并重新开始游戏；如果是拒绝和棋，则当前玩家开始保持请求前的状态。

7.3　制作五子棋游戏的概要设计文档

概要设计是对整个游戏的功能进行概念性的设计，概括地说就是设计如何实现“游戏功能”这个问题。

1．引言

1.1　编写目的

为了让各个开发人员明白五子棋游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。
- 本游戏项目是某公司整个网络游戏系统中的一部分。

2．术语

- 三三禁手：指黑方同时出现两个的三子相连的状态，黑方直接判定为输。
- 四四禁手：指黑方同时出现两个的四子相连的状态，黑方直接判定为输。
- 先手：指执黑一方，先开始第一步。
- 后手：指执白一方，在黑方落子后开始第一步。
- Microsoft：指美国微软公司。
- Intel：指美国的英特尔公司。
- Pentium：是英特尔公司的产品名称。

3．参考文献

《五子棋游戏需求分析说明书》；

《五子棋游戏操作界面设计文档》；

《五子棋游戏的网络连接设计文档》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高鼠标操作易用性，减少键盘的操作。
- 能够支持网络交互，实现两人对战。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。

- Microsoft® Visual C++ 6.0。

4.3　需求概述

（1）能够实现五子棋游戏中的全部规则

- 能够对连五的胜负方进行判断，并给出提示（连五如图 7.4 所示）。
- 能对黑方禁手进行判断（三三禁手和四四禁手，如图 7.5 所示），并能够在落子后进行直接判负的提示。

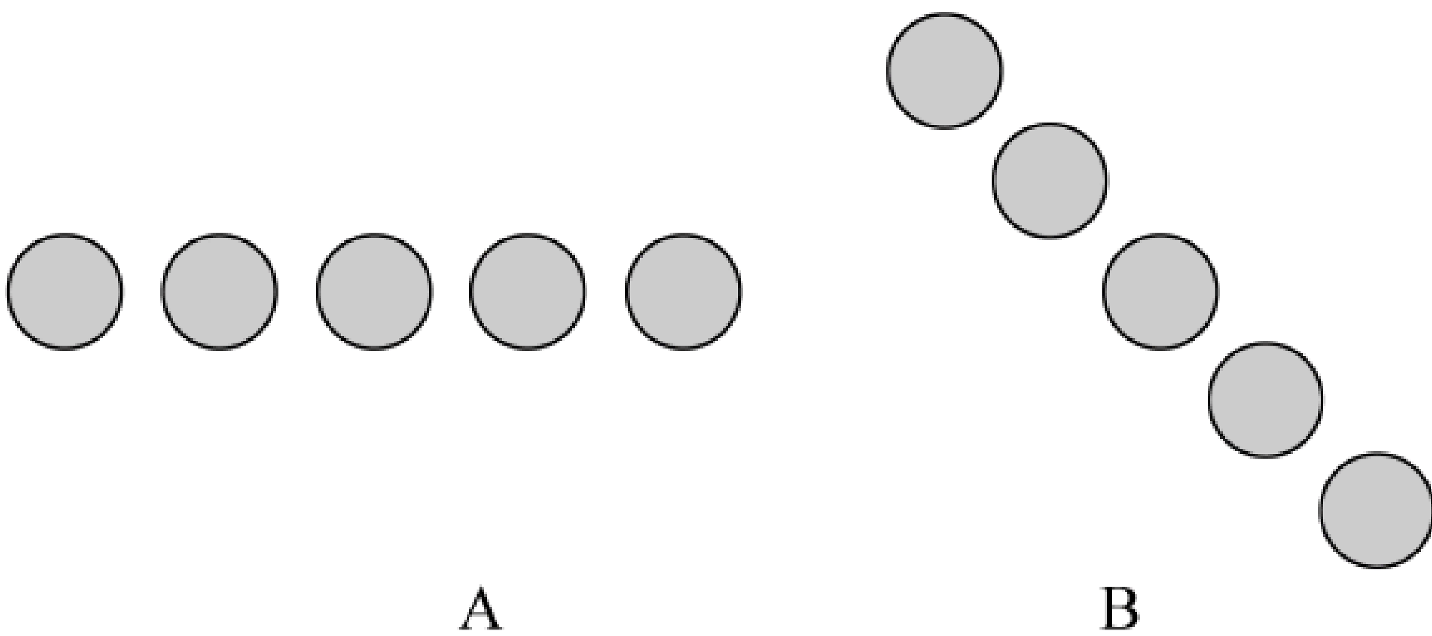


图 7.4　连五

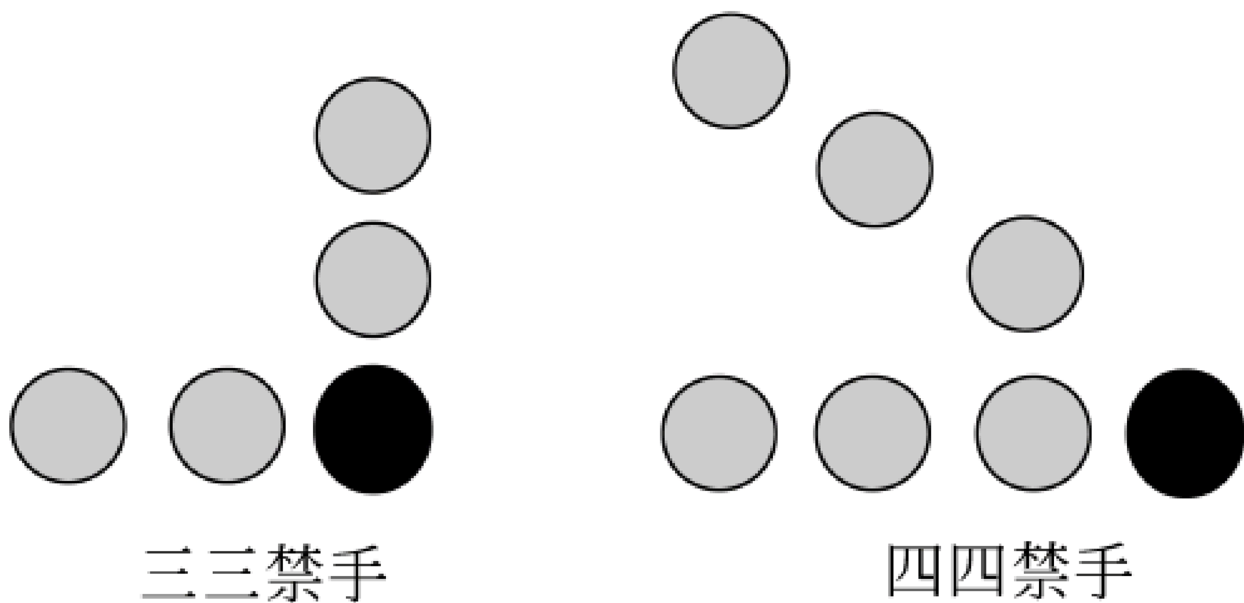


图 7.5　三三禁手和四四禁手

（2）能够支持设置网络端口

- 玩家可以自行设定连接的服务器 IP 地址。通信端口默认使用 5005。

（3）能够支持网络对战

- 支持两个玩家进行联机网络对弈模式。同一时间只能由一方进行游戏，另一方等待，与真实的五子棋游戏相同。
- 支持和棋操作。和棋操作的过程为：首先由玩家向对方玩家发送和棋请求，然后由对方玩家决定是否允许和棋。如果对方玩家允许和棋，就直接给出提示，并重新开始游戏；如果是拒绝和棋，则当前玩家开始保持请求前的状态。

4.4　条件与限制

无。

5．总体设计

5.1　系统功能架构，如图 7.6 所示。

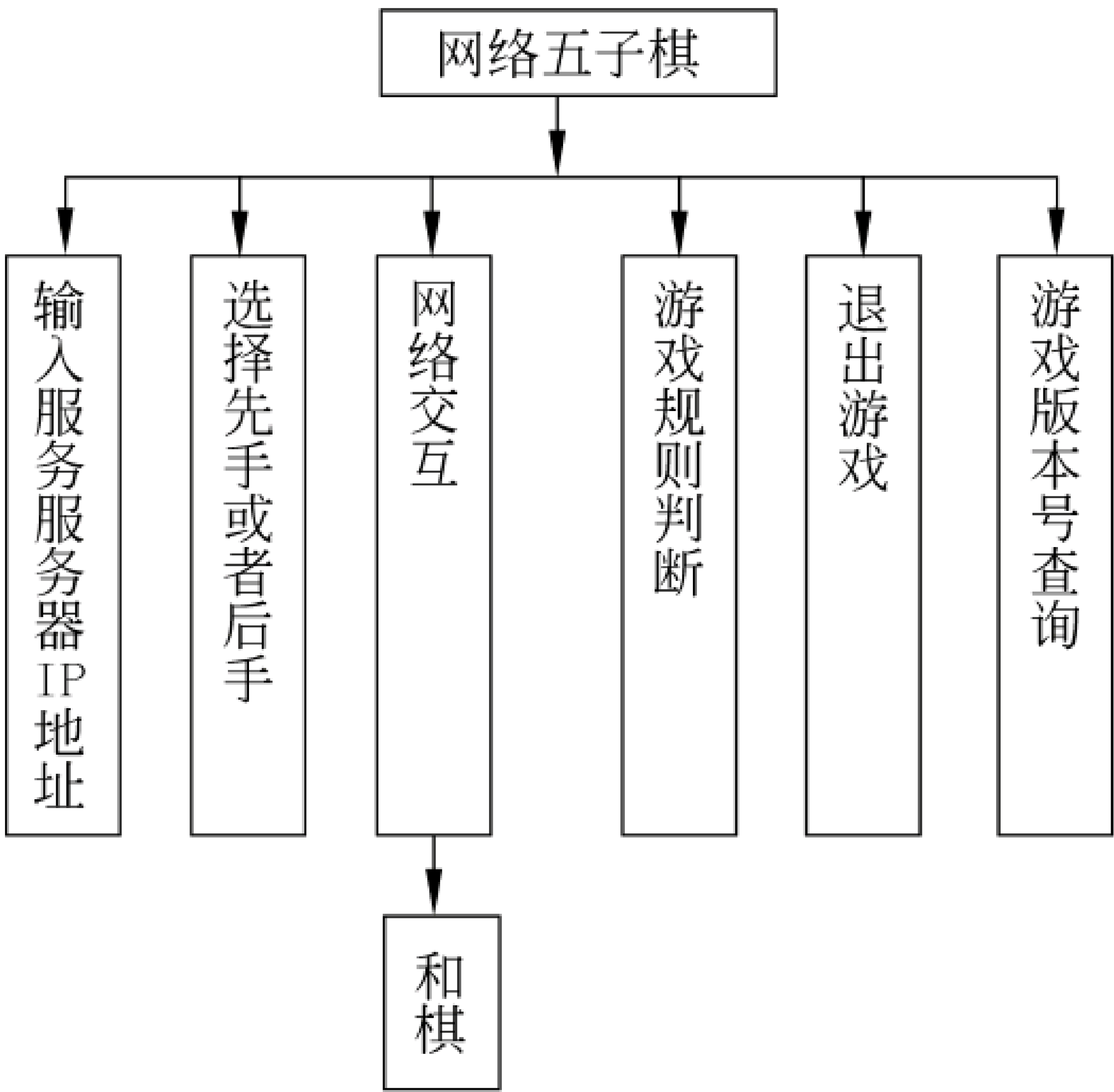


图 7.6　系统功能架构

5.2　处理流程

整个五子棋游戏的处理流程如图 7.7 所示。

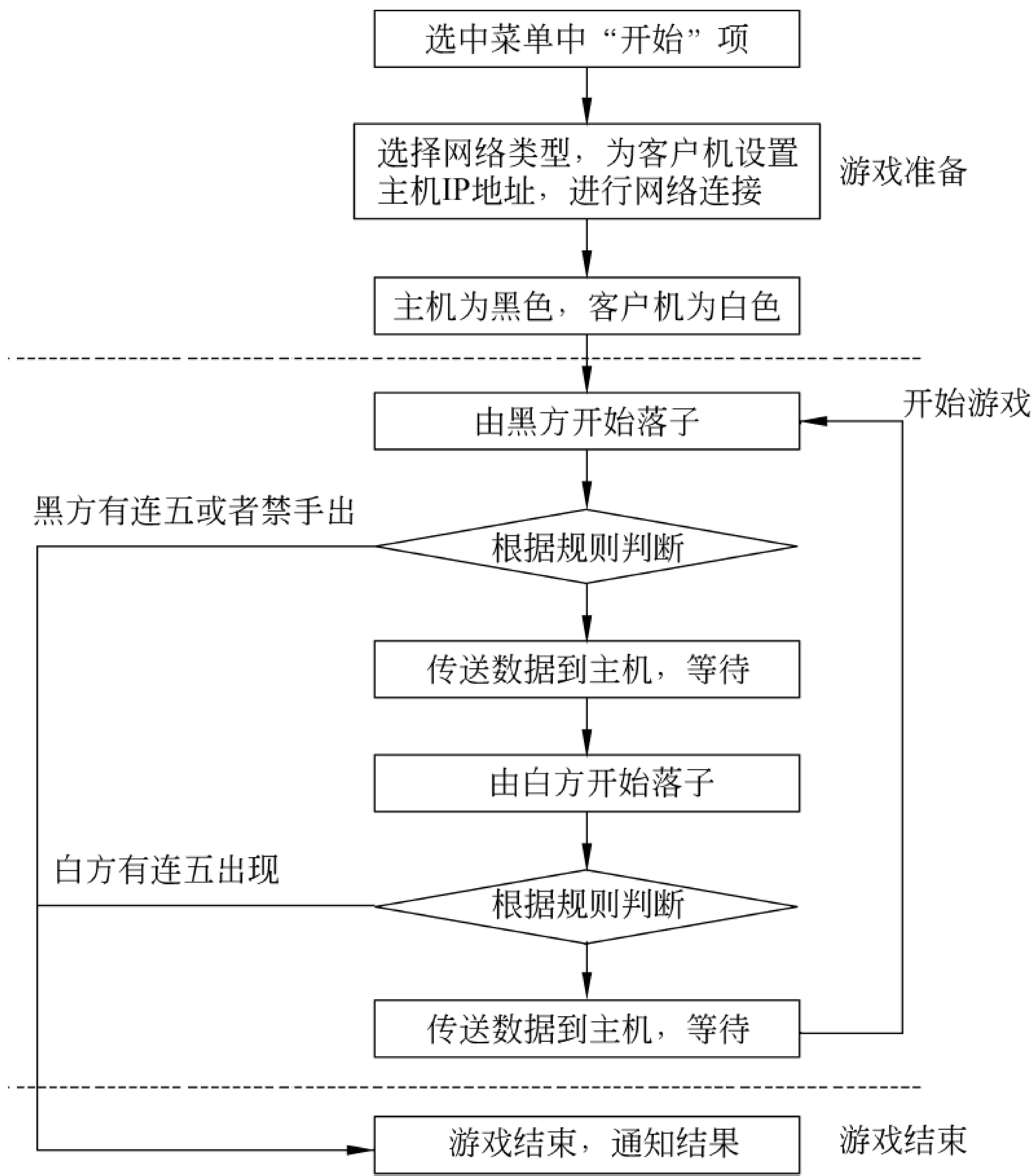


图 7.7　五子棋游戏处理流程

6．接口设计

内容参见《五子棋游戏操作界面设计文档》和《五子棋游戏的网络连接设计文档》。

7．类结构设计

游戏由 5 个类组成，如图 7.8 所示。

- 游戏规则类：主要负责各种类的调用及游戏规则的实现。
- 棋盘窗口类：主要负责棋盘和棋子等的更新和显示。
- 设置对话框类：主要负责参数的设置与连接。
- 网络通信类：主要负责游戏的网络通信。
- 网络协议类：主要负责游戏网络通信协议的实现。

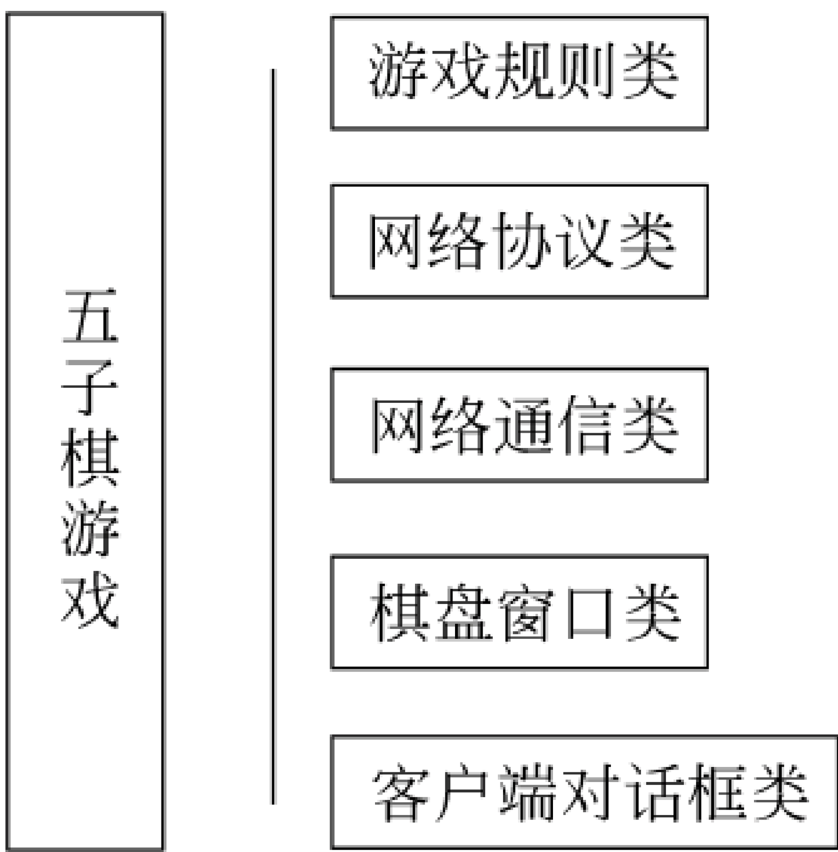


图 7.8　游戏主要类结构

8．出错处理设计

8.1　出错输出信息

当游戏中的一方出现错误，例如网络通信中断、程序出错等，游戏的错误处理类会采用弹出对话框的方式来提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误，正常一方采用中止当前游戏，并重新开始新游戏的方法来处理游戏中的错误。

9．维护设计

由于整个五子棋游戏项目在开发完成后，基本不会有太多的变动。所以维护主要的任务是把用户使用中的错误解决。

7.4　五子棋游戏的操作界面设计文档

游戏界面设计文档，是在游戏开发中特有的一种文档格式，主要是采用图文的方式来描述在游戏中的操作界面。通过这个文档也使项目的提出者能够大概知道游戏实现后，其游戏操作界面是否符合用户的要求。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确游戏的操作界面是如何设计的，特制定本文档来用于描述本公司五子棋游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。
- 本游戏项目是某公司整个网络游戏系统中的一部分。

2．文档范围

包含本公司五子棋游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《五子棋游戏需求分析说明书》。

5．游戏界面设计

五子棋游戏中的菜单设计如图 7.9 所示。在五子棋游戏中，当用户选择“操作”|“开始游戏”命令后，就会弹出网络设置对话框，其设计如图 7.10 所示。



图 7.9　五子棋游戏的菜单

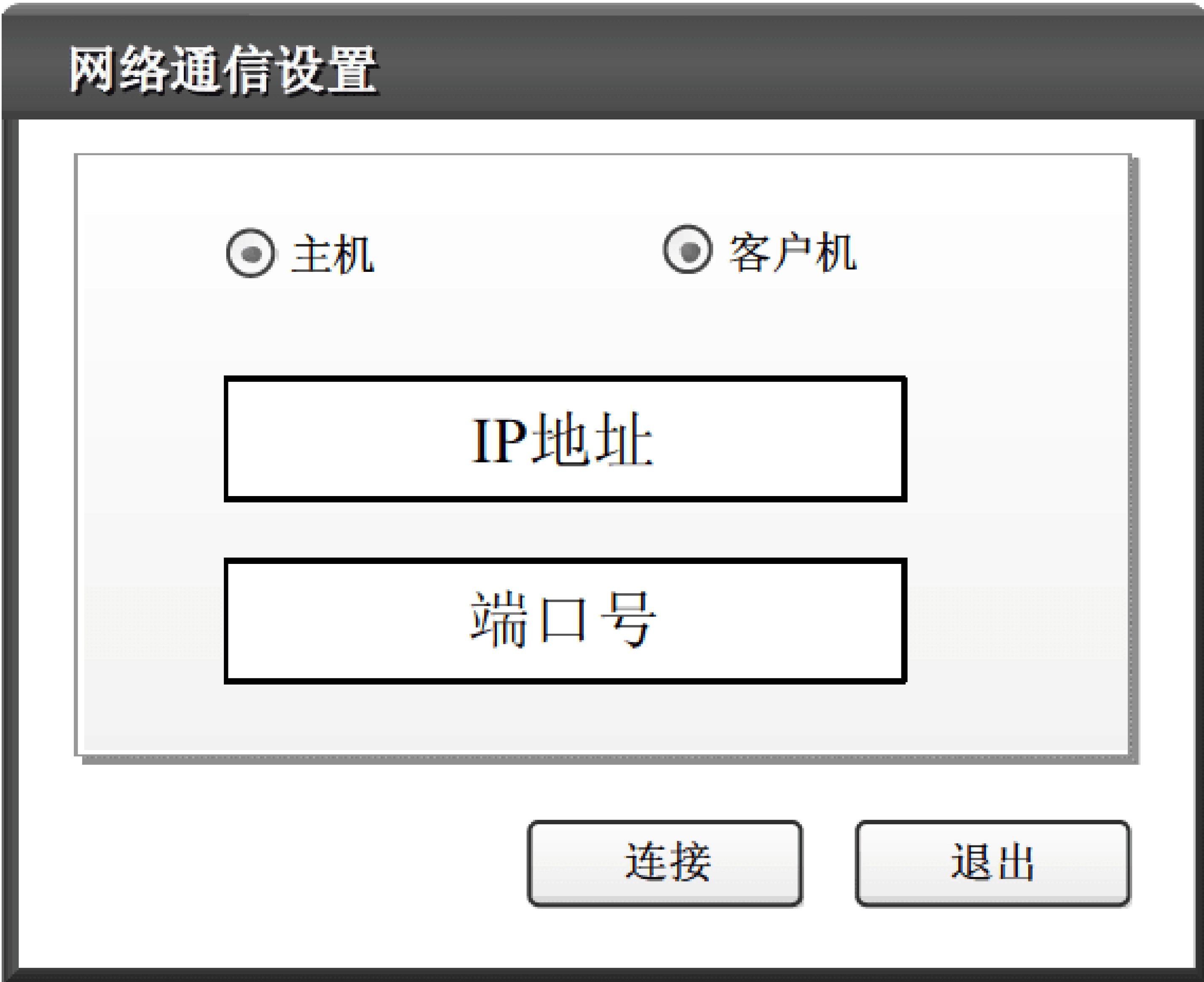


图 7.10　网络通信设置对话框

当连接成功后，用户就可以开始进行网络五子棋博弈。游戏主界面的布局和设计如图 7.11 所示。

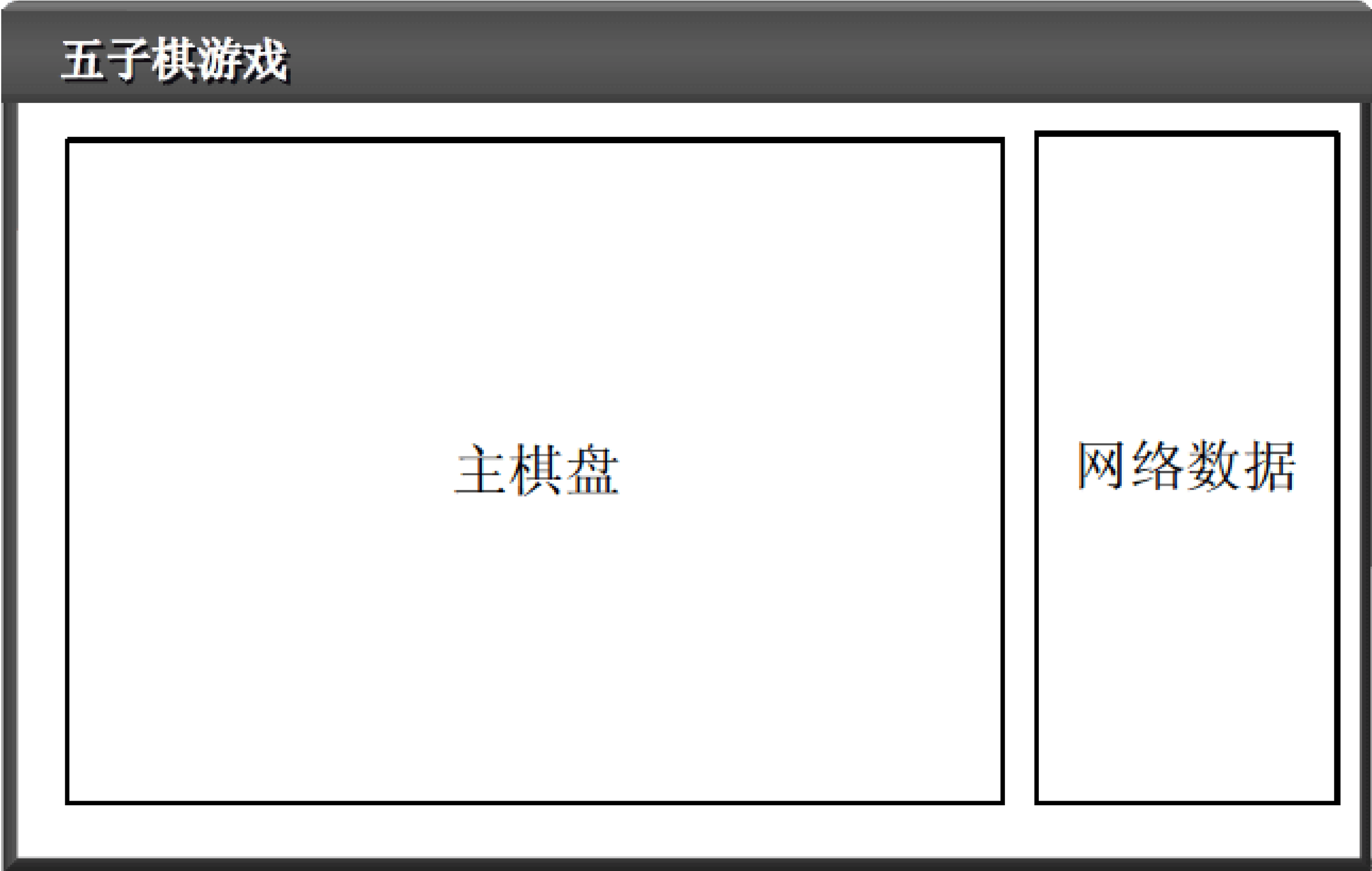


图 7.11　五子棋游戏主界面布局设计

7.5　总　　结

通过本章的学习，希望各位读者可以掌握游戏项目是如何进行前期工作的。

（1）进行需求分析，并编写相关报告。

（2）项目的概要设计文档的编写。

（3）游戏项目特有的操作界面设计文档的编写。

注意，这些工作中，最重要的一步是如何进行需求分析。因为只有好的需求分析，才能够设计出用户满意的软件。如果需求分析做得不好，那么游戏设计的风险就越大，最糟糕的结果是有可能导致项目流产。读者通过该示例的学习，能够自己编写项目需求分析说明书。

在第 8 章中，笔者将开始五子棋游戏的详细设计，在掌握好本章的内容后，继续我们的游戏项目开发之路吧。

第 8 章　五子棋游戏界面与通信开发详解

本章是在第 7 章的基础上，对游戏项目开发中的详细设计、测试实例文档进行讲解，并开始游戏界面的代码及网络连接的代码实现。

本章主要涉及的内容如下：

- 游戏的详细设计：通过实例文档让读者知道如何进行游戏的详细设计。
- 网络协议的实现：主要是通过实例设计文档和代码来学习网络协议是如何实现的。
- 游戏交互界面的实现：主要是对详细设计文档中描述的内容进行代码实现。

8.1　五子棋游戏的详细设计

在这一节中，读者将了解到五子棋游戏项目开发中的详细设计。在游戏开发的详细设计中，需要对各个功能函数、数据结构或者类进行详细的描述。不仅如此，还需要给出实现各个功能的详细流程图。

为了方便讲解和阅读，笔者不再采用前面的那种实例文档的方式进行讲解，而采用分步突出的方法来讲解。因为这样才能更加清楚其中的重点内容。

8.1.1　五子棋游戏详细设计的目标

五子棋游戏详细设计的目标是为了描述如何实现五子棋游戏项目需求分析得到的客户功能要求。不仅如此，制作详细设计的目标还是为了指导后续的编码工作。明确地说，就是将某一位软件设计师的设计拿给不同的程序员去写代码,写出来的程序处理流程相同。这样可以使后续的评审或者当其他人接替这项工作时，只需查看详细设计文档，就能明白软件设计师当时是怎样设计这些流程的。

8.1.2　五子棋游戏功能结构及名称定义

在前面的 7.4 节中已经列举了五子棋游戏所要包含的全部功能结构，但只是对其功能结构进行了概要性的描述。而在详细设计中，将用一系列图表列出五子棋游戏的软件系统内的每个程序（包括每个类和函数）的名称、标识符及其之间的层次结构关系。

五子棋游戏中各个类的名称及关系，如图 8.1 所示。

图中各个类的功能说明如下。

- 游戏规则类：主要负责游戏规则的实现。
- 棋盘窗口类：主要负责棋盘和棋子等的更新和显示。
- 设置对话框类：主要负责服务器及客户端参数的设置与连接。
- 网络通信类：主要负责游戏的网络通信。
- 网络协议类：主要负责游戏网络通信协议的实现。

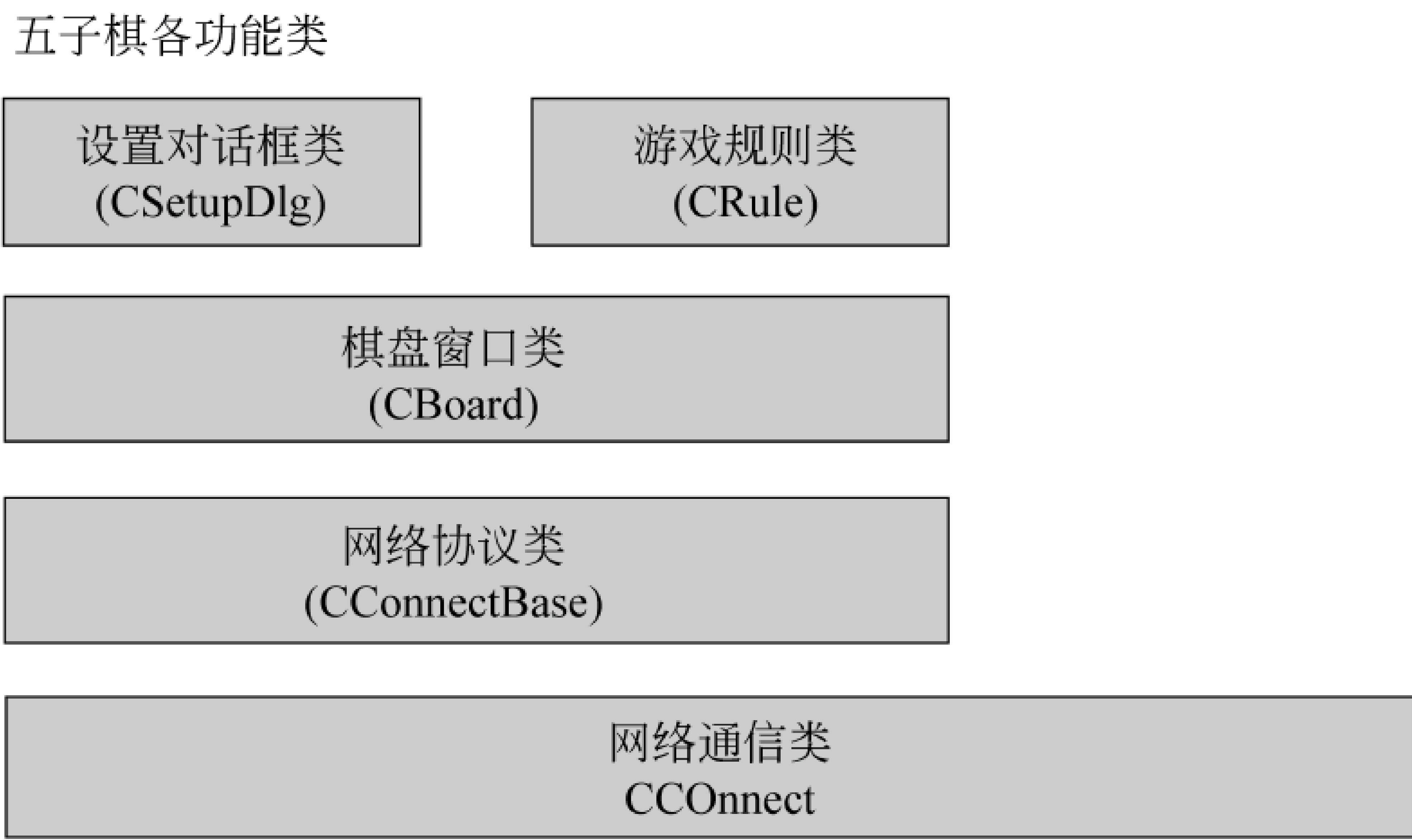


图 8.1　五子棋游戏功能结构汇总

8.2　网络通信协议类的设计与实现

网络通信协议类，是整个五子棋游戏的核心类之一，其主要负责进行游戏数据的封包、解包工作。

8.2.1　网络通信协议的设计

分析五子棋游戏需求后可以得出，网络传输的数据有如下 3 种类型：

- 游戏中棋子在棋盘中的坐标，即棋子的位置。
- 游戏中的控制信息，例如“和棋”。
- 其他扩展信息，例如，以后游戏中需要增加聊天功能等。

为了能适应这 3 种不同种类的信息的传输，所以在游戏的网络传输协议中，必须加入一个网络协议包结构。其数据域格式如图 8.2 所示。

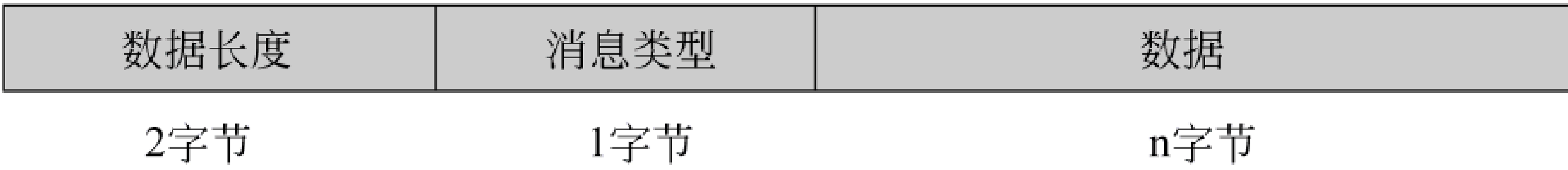


图 8.2　网络协议结构体

注意其中数据长度是包含本身长度字段 2 个字节的。例如，如果数据长度为 4 个字节，那么长度字段必须是 7，而不是 4。

8.2.2　各种数据类型的详细格式

本节将对不同数据类型的网络传输数据包进行详细说明。

1. 棋子位置数据包

棋子位置数据包是网络传输中的主要数据，在这里把棋盘当成二维数组。所以在网络中实际传输的是棋子在二维数组中的行列号。其详细格式如表 8.1 所示。

表 8.1　棋子位置数据格式

数 据 域 名	长　　度	真 实 数 据	备　　注
数据长度	2	0X06	
消息类型	1	0X01	棋子位置类型
数据	2	0X00	棋子的行序号（根据实际数据变化）
		0X00	棋子的列序号（根据实际数据变化）
		0X00	棋子颜色（0 黑色，1 白色）

2. 游戏控制数据中的“和棋”数据包

游戏控制数据中的“和棋”数据包是指游戏中进行“和棋”操作时，向对方发送的网络传输数据包。

（1）“和棋请求”数据格式如表 8.2 所示。

表 8.2　和棋请求数据格式

数 据 域 名	长　　度	真 实 数 据	备　　注
数据长度	2	0X03	
消息类型	1	0X02	游戏控制中和棋

（2）“同意和棋回应”数据格式如表 8.3 所示。

表 8.3　同意和棋回应数据格式

数 据 域 名	长　　度	真 实 数 据	备　　注
数据长度	2	0X03	
消息类型	1	0X03	同意和棋回应

（3）“拒绝和棋回应”数据格式如表 8.4 所示。

表 8.4　拒绝和棋回应数据格式

数 据 域 名	长　　度	真 实 数 据	备　　注
数据长度	2	0X03	
消息类型	1	0X04	拒绝和棋回应

3．其他扩展数据

其他扩展信息，是留给以后游戏扩展使用的，其数据格式如表 8.5 所示。

表 8.5　其他扩展数据格式

数据域名	长　度	真实数据	备　注
数据长度	2	N+3	根据实际数据变化
消息类型	1	0X05	0X05 到 0XFF 都是扩展的消息类型
数据	N		根据需要进行扩展

8.2.3　网络通信协议的实现

为了满足网络通信的需要，已经制定了各种通信需要的数据结构，现在就来看看如何用 C++代码来实现这些数据结构。其代码如代码 8.1 所示。

代码 8.1　通信需要使用的数据结构定义

```
01  #ifndef __CONNECT_DATA_H__
02  #define __CONNECT_DATA_H__
03  //落子消息
04  #define MSG_PUTSTEP     0x01
05  //和棋请求消息
06  #define MSG_DRAW        0x02
07  //同意和棋消息
08  #define MSG_AGREE_DRAW  0x03
09  //拒绝和棋消息
10  #define MSG_REFUSE_DRAW 0x04
11  //其他消息
12  #define MSG_EXTERN      0x05
13  typedef struct _tagMsgStruct {
14      USHORT len;
15      BYTE msgType;           //消息 ID
16      int x;                  //落子信息
17      int y;
18      int color;
19      BYTE byMsg[128];        //其他消息内容
20  } MSGSTRUCT;20
21  #endif
```

8.3　交互界面的设计与实现

优秀的交互界面是提高游戏满意度的一种方式。在 Windows 操作系统中交互界面分为对话框和菜单两种，在游戏设计上也同样如此。交互界面的友好性是决定游戏能否吸引用户的要素之一。

8.3.1　控制菜单的设计

经过前面的需求分析后，可以得出：在五子棋游戏中，有如下 3 种操作命令需要使用菜单来支持。

（1）新游戏操作，当用户进入主界面后，需要这个菜单项来开始新的游戏。开始新游戏之前需要对网络进行相应设置。

（2）和棋操作，在游戏中由一方用户提出用于提前结束游戏，相当于提出方认输，并重新开始新一轮的游戏。

（3）退出游戏操作，在当前用户不需要再玩游戏时，直接退出整个游戏界面。

开发的这款五子棋游戏是在 Windows 上运行的，为了保持与系统界面的一致性，这里采用标准 Windows 菜单进行设计，如图 8.3 所示。

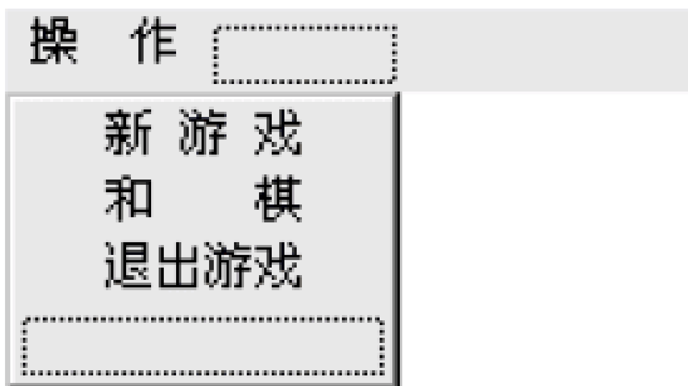


图 8.3　五子棋游戏菜单

8.3.2　控制菜单的实现

为了实现图 8.3 所示的菜单，需要在五子棋项目（FiveChess）的资源中加入一个菜单栏（IDR_MAIN_MENU），并按照表 8.6 所示的设置菜单中菜单栏的对应 ID 资源号。

表 8.6　菜单ID资源

ID	菜 单 名 称
ID_NEW_GAME_MENU	新游戏
ID_DRAW_GAME_MENU	和棋
ID_EXIT_GAME_MENU	退出游戏

设置完菜单中的菜单栏 ID 后，还需要使用类向导，如图 8.4 所示，用来给各菜单栏添加单击响应函数。

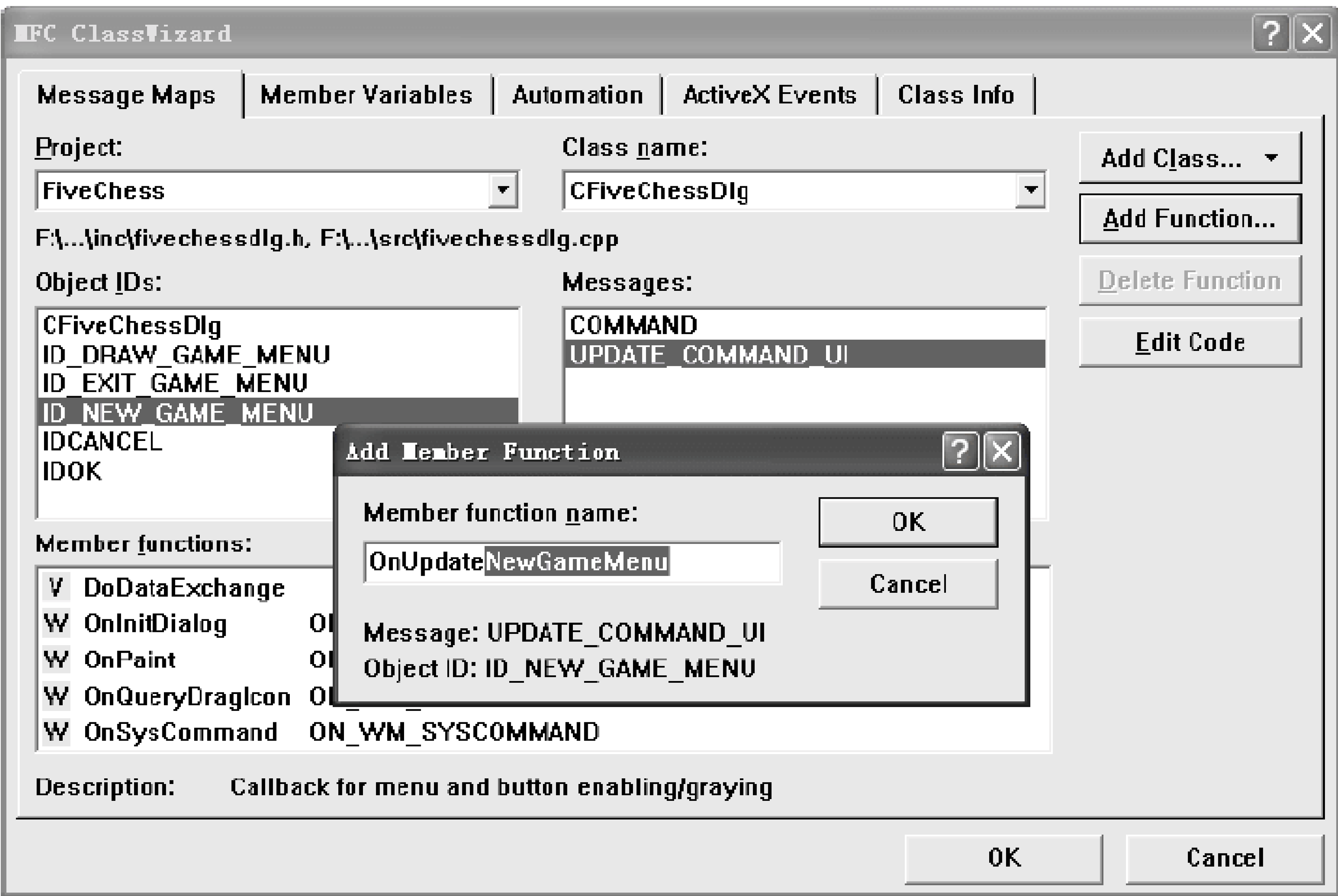


图 8.4　使用类向导给菜单栏添加响应函数

为了让初学者知道什么是类向导及如何使用，笔者将详细讲解如何使用类向导给

ID_NEW_GAME_MENU 添加响应的方法，如下所述。

（1）启动类向导，其快捷键为 Ctrl+W。

（2）选中 ClassName 为 CfiveChessDlg。

（3）在 Object IDs 中选中想要添加响应函数的菜单 ID_NEW_GAME_MENU。

（4）在 Messages 中选中 UPDATE_COMMAND_UI 选项。

（5）单击 AddFunction 按钮。

（6）在弹出的 Add Member Function 对话框中输入响应函数名称，一般采用默认值。

（7）单击 OK 按钮完成函数的添加。

在给每个菜单栏添加完响应函数后，可以单击 Edit Code 按钮来编辑相关响应函数的源代码，类向导自动添加完成后的内容如代码 8.2 所示。

代码 8.2　菜单响应函数

```
01  // FiveChessDlg.h 类声明头文件
02  #if !defined(AFX_FIVECHESSDLG_H__)
03  #define AFX_FIVECHESSDLG_H__
04
05  ////////////////////////////////////////////////////////////////////
06  // CFiveChessDlg 对话框类的定义
07
08  class CFiveChessDlg : public CDialog
09  {
10  public:
11      CFiveChessDlg(CWnd* pParent = NULL);    //构造函数
12
13      enum { IDD = IDD_FIVECHESS_DIALOG };    //资源与类连接
14
15  protected:
16      virtual void DoDataExchange(CDataExchange* pDX);
17
18  protected:
19      HICON m_hIcon;                          //图标对象
20      CMenu m_main_menu;                      //主菜单对象
21      virtual BOOL OnInitDialog();            //初始化函数声明
22      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
23      afx_msg void OnPaint();                 //绘图函数声明
24      afx_msg HCURSOR OnQueryDragIcon();
25                                        //菜单栏 NEW_GAME 的响应函数声明
26      afx_msg void OnUpdateNewGameMenu(CCmdUI* pCmdUI);      afx_msg
27                                        //菜单栏 EXIT_GAME 的响应函数声明
28      void OnUpdateExitGameMenu(CCmdUI* pCmdUI);
29                                        //菜单栏 DRAW_GAME 的响应函数声明
30      afx_msg void OnUpdateDrawGameMenu(CCmdUI* pCmdUI);
31      DECLARE_MESSAGE_MAP()
32  };
33  #endif // !defined(AFX_FIVECHESSDLG_H__)
34
35  // FiveChessDlg.cpp 实现文件开始
36
37  #include "stdafx.h"                     //插入头文件
38  #include "FiveChess.h"
39  #include "FiveChessDlg.h"
40  ////////////////////////////////////////////////////////////////////
41  // CFiveChessDlg 对话框类的实现
```

```
42
43  CFiveChessDlg::CFiveChessDlg(CWnd* pParent /*=NULL*/)
44      : CDialog(CFiveChessDlg::IDD, pParent)
45  {
46      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);     //加载图标资源
47  }
48
49  void CFiveChessDlg::DoDataExchange(CDataExchange* pDX)
50  {
51      CDialog::DoDataExchange(pDX);               //调用父类的数据加载函数
52  }
53
54  BEGIN_MESSAGE_MAP(CFiveChessDlg, CDialog)   //响应函数映射关系表
55      ON_WM_SYSCOMMAND()
56      ON_WM_PAINT()                           //绘图函数声明
57      ON_WM_QUERYDRAGICON()
58                                              //NEW_GAME 映射函数关系
59      ON_UPDATE_COMMAND_UI(ID_NEW_GAME_MENU, OnUpdateNewGameMenu)
60                                              //EXIT_GAME 映射函数关系
61      ON_UPDATE_COMMAND_UI(ID_EXIT_GAME_MENU, OnUpdateExitGameMenu)
62                                              //DRAW_GAME 映射函数关系
63      ON_UPDATE_COMMAND_UI(ID_DRAW_GAME_MENU, OnUpdateDrawGameMenu)
64  END_MESSAGE_MAP()
65  BOOL CFiveChessDlg::OnInitDialog()          //初始化函数实现
66  {
67      CDialog::OnInitDialog();                //调用父类的初始化函数
68                                              //加载系统菜单
69      CMenu* pSysMenu = GetSystemMenu(FALSE);
70      if (pSysMenu != NULL)
71      {
72          CString strAboutMenu;               //关于菜单字符串
73          strAboutMenu.LoadString(IDS_ABOUTBOX);
74          if (!strAboutMenu.IsEmpty())
75          {
76              pSysMenu->AppendMenu(MF_SEPARATOR);
77              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutM-
                enu);
78          }
79      }
80      SetIcon(m_hIcon, TRUE);                 //设置大图标
81      SetIcon(m_hIcon, FALSE);                //设置小图标
82
83      m_main_menu.LoadMenu(IDR_MAIN_MENU);    //加载主菜单资源
84
85      SetMenu(&m_main_menu);                  //设置主菜单
86
87      return TRUE;                            //返回真，初始化成功
88  }
89  ...                                         //省略部分代码，请查看光盘实例
90                                              //NEW_GAME 响应函数实现
91  void CFiveChessDlg::OnUpdateNewGameMenu(CCmdUI* pCmdUI)
92  {
93      if(IDOK==m_setup_dlg.DoModal())         //调用网络设置对话框
94      {
95          NewGameStart(m_setup_dlg.m_isHost);
                                            //单击确定按钮，调用开始新游戏函数
96      }
97  }
```

```
98                                              //EXIT _GAME 响应函数实现
99  void CFiveChessDlg::OnUpdateExitGameMenu(CCmdUI* pCmdUI)
100 {
101     SetMenu(NULL);                              //清空主菜单对象
102
103     Cdialog::OnCancel();                        //调用父类对话框的退出函数
104 }
105                                                 //DRAW _GAME 响应函数实现
106 void CFiveChessDlg::OnUpdateDrawGameMenu(CCmdUI* pCmdUI)
107 {
108     DrawGame();                                 //调用和棋函数
109 }
```

代码解析：第 95 行是新游戏菜单栏的响应函数实现，主要通过调用 NewGameStart()函数来实现开始新游戏。第 108 行是和棋菜单栏的响应函数实现，主要通过调用 DrawGame()函数来实现发送和棋包给对手，如果对方同意，则进行和棋。

技巧： *在基于对话框模式的 MFC 程序中，系统不会自动加载菜单栏，需要用户动态地加载。*

8.3.3　网络设置对话框的设计

从需求分析得知，当用户开始新游戏时，需要对网络进行设置。其包含的内容如下：

（1）可以选择当前用户是主机，还是客户机。如果是主机则执黑子，如是客户机则执白子。

（2）当用户选择为主机时，IP 地址默认填写为“127.0.0.1”，端口号由用户填写。

（3）当用户选择为客户机时，则需要再设置连接到的主机 IP 地址及端口号。

（4）如果前面已经设置过，则重新单击“新游戏”选项时，把上一次设置的 IP 地址和端口号显示出来。

为了满足上面的需求，笔者设计了一个对话框界面如图 8.5 所示，并且把这个对话框资源同 CsetupDlg 类相关联起来。同时，因为在退出游戏后，还要把上一次设置的记录进行保存，所以程序中必须还可以操作文件。为了方便，这里笔者使用了 INI 系统配置文件来完成相关任务。

图中的各资源对应 ID 号及名称如表 8.7 所示。



图 8.5　网络设置对话框

表 8.7　网络设置对话框的资源ID及名称

ID	显 示 名 称	资 源 类 型
IDC_HOST_OPTION	主机	单选项
IDC_CLIENT_OPTION	客户机	单选项
IDC_IP_ADDRESS_EDIT	无	IP 地址编辑框
IDC_NET_PORT_EDIT	无	文本编辑框
IDOK	确认	按钮
IDCANCEL	取消	按钮
IDC_STATIC	主机类型选择	静态标签

8.3.4　网络设置对话框的实现

根据前面要求设置相关 ID 资源和名称后，就需要使用类向导来添加响应函数和成员变量。前面已经介绍过添加响应函数的方法，添加成员变量的方法与其类似，这里就不再复述。对话框实现如代码 8.3 所示。

代码 8.3　网络设置对话框的实现

```
01  #if !defined(AFX_SETUPDLG_H__)
02  #define AFX_SETUPDLG_H__
03
04  // SetupDlg.h 网络设置对话框类的头文件
05
06                                         //声明设置对话框类
07  class CSetupDlg : public CDialog
08  {
09  public:
10      CSetupDlg(CWnd* pParent = NULL);//构造函数
11      enum { IDD = IDD_SETUP_DLG };
12      CIPAddressCtrl  m_ip_addr;         //IP 地址输入编辑框控件对象
13      UINT    m_net_port;                //端口号
14  public:
15      BOOL m_isHost;                     //主机类型，TRUE:主机 FALSE:客户机
16
17      protected:
18      virtual void DoDataExchange(CDataExchange* pDX);
19  protected:
20                                         //消息与响应函数映射关系
21      virtual void OnOK();               //确认键响应函数
22      virtual void OnCancel();           //取消键响应函数
23      afx_msg void OnClientOption();     //选中客户机选项响应
24      afx_msg void OnHostOption();       //选中主机选项响应
25      DECLARE_MESSAGE_MAP()
26  };
27  #endif
28
29  // SetupDlg.cpp 网络设置对话框类的实现文件
30
31
32  #include "stdafx.h"                    //插入头文件
33  #include "FiveChess.h"
34  #include "SetupDlg.h"                  //插入类声明头文件
35  //////////////////////////////////////////////////////////////
36  // CSetupDlg 对话框类的实现
37  CSetupDlg::CSetupDlg(CWnd* pParent /*=NULL*/)
38      : CDialog(CSetupDlg::IDD, pParent)
39  {                                      //构造函数
40      m_net_port = 0;                    //初始化成员
41  }
42
43  void CSetupDlg::DoDataExchange(CDataExchange* pDX)
44  {
45      CDialog::DoDataExchange(pDX);      //成员与控件关联
46      DDX_Control(pDX, IDC_IP_ADDRESS_EDIT, m_ip_addr);
```

```
47      DDX_Text(pDX, IDC_NET_PORT_EDIT, m_net_port);
48      DDV_MinMaxUInt(pDX, m_net_port, 1, 65000);
49  }
50
51  BEGIN_MESSAGE_MAP(CSetupDlg, CDialog)   //响应函数与控件关联
52      ON_BN_CLICKED(IDC_CLIENT_OPTION, OnClientOption)
53      ON_BN_CLICKED(IDC_HOST_OPTION, OnHostOption)
54  END_MESSAGE_MAP()
55
56  //////////////////////////////////////////////////////////////
57  // CsetupDlg 类的消息响应函数实现
58
59  void CSetupDlg::OnOK()                  //单击确认按钮响应
60  {
61      CString strIP;                      //定义临时字符串变量
62      Cstring strPort;
63      UpdateData(TRUE);                   //更新显示到变量
64      m_ip_addr.GetWindowText(strIP);     //得到 IP 地址编辑框中输入的字符
65      strPort.Format("%d", m_net_port);   //格式化端口号为字符串
66      if(m_isHost)                        //判断是否选择了主机选项
67      {                                   //把主机的 PORT 写入配置文件
68          WritePrivateProfileString("HOST", "PORT", strPort, ".\\config.
            ini");
69      }
70      else
71      {                                   //把客户机的 IP 和 PORT 写入配置文件
72          WritePrivateProfileString("CLIENT", "IP", strIP, ".\\config.
            ini");
73          WritePrivateProfileString("CLIENT", "PORT", strPort, ".\\con-
            fig.ini");
74      }
75      CDialog::OnOK();                    //调用父类成员函数
76  }
77
78  void CSetupDlg::OnCancel()              //单击取消按钮响应函数
79  {
80      CDialog::OnCancel();                //调用父类成员函数
81  }
82
83  void CSetupDlg::OnClientOption()        //响应选中客户机选项
84  {
85      char str[128] = {0};
86      m_ip_addr.EnableWindow(TRUE);
87                                          //读配置文件中的客户机 IP 地址
88      GetPrivateProfileString("CLIENT", "IP", "", str, 127, ".\\config.
        ini");
89      m_ip_addr.SetWindowText(str);
90      memset(str,0, 128);
91                                          //读配置文件中的客户机端口号
92      GetPrivateProfileString("CLIENT", "PORT", "5000", str, 127, ".\\co-
        nfig.ini");
93      m_net_port = atoi(str);
94      m_isHost = FALSE;                   //设置连接类型变量
95      UpdateData(FALSE);                  //把变更数据更新显示
96  }
97
98  void CSetupDlg::OnHostOption()          //响应选中主机选项
99  {
```

```
100     char str[128] = {0};
101     m_ip_addr.EnableWindow(FALSE);      //使 IP 输入编辑框无效
102     GetPrivateProfileString("HOST", "IP", "", str, 127, ".\\config.
        ini");
103     m_ip_addr.SetWindowText(str);
104     memset(str,0, 128);
105     GetPrivateProfileString("HOST", "PORT", "5000", str, 127, ".\\con-
        fig.ini");
106     m_net_port = atoi(str);
107     m_isHost = TRUE;
108     UpdateData(FALSE);
109 }
110 BOOL CSetupDlg::OnInitDialog()          //初始化对话框函数
111 {
112     CDialog::OnInitDialog();
113                                         //设置默认为选中主机选项
114     ((CButton*)GetDlgItem(IDC_HOST_OPTION))->SetCheck(1);
115     OnHostOption();
116     return TRUE;                        //返回真，表示初始化成功
117 }
```

代码解析：第 68 行当用户单击设置对话框中的“确定”按钮后，就会把 IP 和端口号写入到 config.ini 配置文件中。第 88 行在下一次用户再进入程序时，又会读取配置文件中的数据填充到对话框显示中，这样就实现了保存设置的功能。

8.4　总　　结

通过本章的学习，希望各位读者可以掌握游戏开发项目中的如下内容：

- ❑ 明白详细设计的目标和如何编写详细设计。
- ❑ 掌握五子棋游戏项目中的网络通信类是如何设计和实现的。
- ❑ 游戏的交互界面的设计与实现。

以上内容只是详细设计的一小部分，在第 9 章中，将更加深入地讲解五子棋游戏项目开发的核心内容。包括如下内容：

- ❑ 棋盘类的设计与实现。
- ❑ 网络交互算法的设计与实现。
- ❑ 游戏规则类的设计与实现。

游戏开发中的核心内容一般都比较复杂，笔者力求用简单的语言来描述这些内容，希望读者能够掌握其精髓。

第 9 章　五子棋游戏的核心算法设计与实现

通过前一章的讲解，已经初步涉及五子棋游戏的代码编写部分。在本章中将继续深入地讲解五子棋游戏项目开发的核心。

本章主要涉及的内容如下：

- 棋盘窗口类的设计与实现：知道如何设计和实现五子棋游戏的棋盘和棋子的显示。
- 网络交互算法的设计与实现：知道如何进行网络交互。
- 游戏规则类的设计与实现：知道如何实现游戏主流程。

9.1　棋盘窗口类的设计与实现

棋盘窗口类主负责显示棋盘和棋子，同时还要处理鼠标输入信息，是整个游戏中的重点，本节也是整个游戏开发中的重点内容。

9.1.1　棋盘窗口类的设计思想

通过分析用户的需求后，可以得出棋盘窗口类应支持如下几个功能：

- 能够显示棋盘和棋子图片。
- 能够接收用户鼠标输入，并把相应的坐标转换成为行列数据填充到棋子数组中，实现棋子的显示。
- 能够处理来自网络通信的各种数据。
- 能够调用规则类对象来判断游戏的胜负。
- 能够支持清空游戏棋盘上的棋子。

下面笔者分 5 个小节来说明，如何设计这些功能。

1．棋盘和棋子的显示

要把棋盘（如图 9.1 所示）和棋子显示出来，分为如下几个过程。

（1）得到当前绘图 DC 句柄，并保存当前绘图环境到内存 DC 中。

（2）载入一张棋盘 BMP 图片，并在主界面对话框的客户区绘画出来。

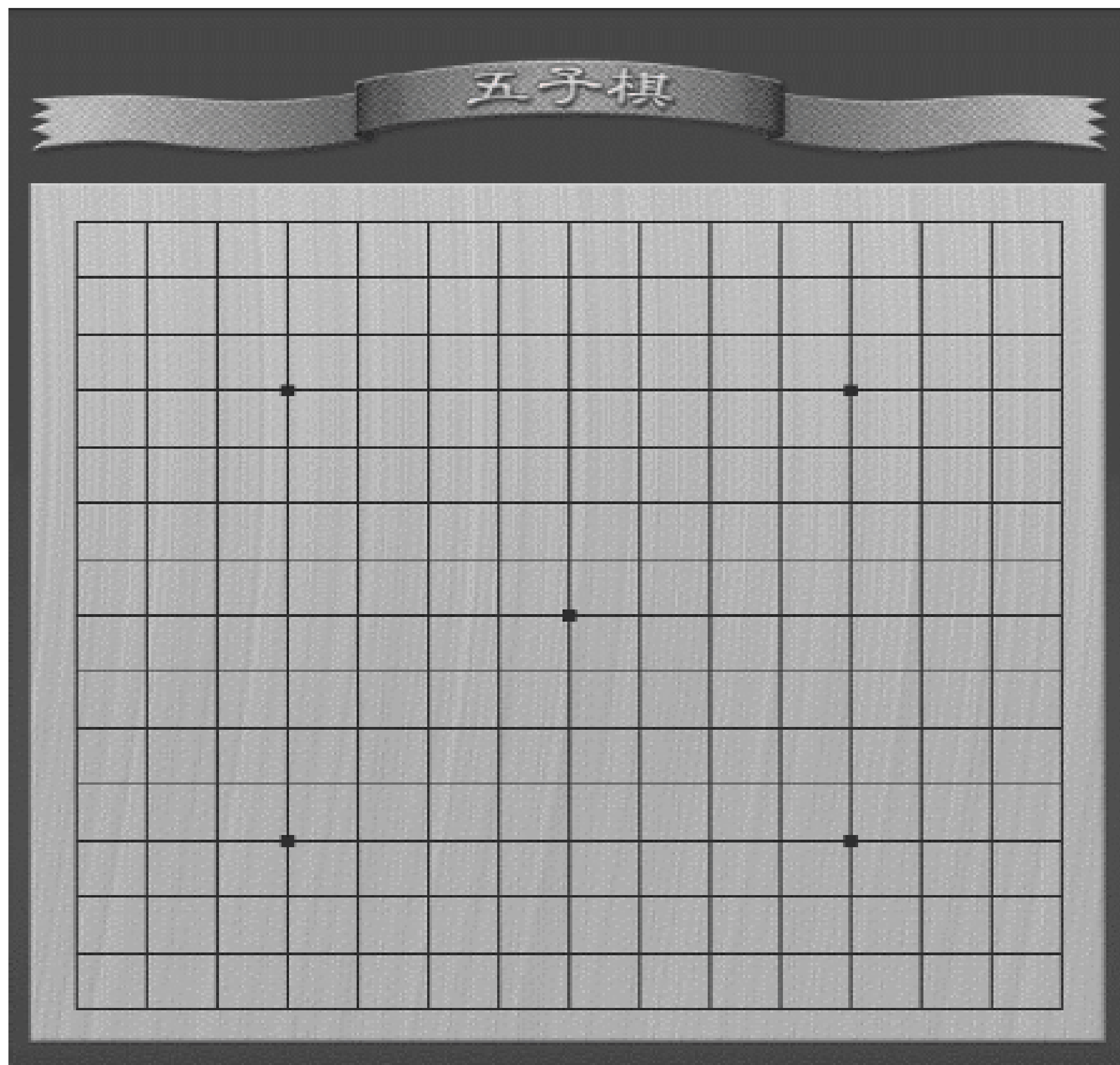


图 9.1　棋盘

（3）遍历棋子数组，根据数组中的数据，把相应颜色的棋子图片绘出来。

（4）更新内存 DC 到当前绘图 DC 中。这样就可以把棋盘和棋子图片显示出来了。

2．鼠标输入数据的处理

要实现鼠标输入数据的处理，可以分为如下几步：

（1）得到鼠标在当前窗口中点击的左键坐标。

（2）根据棋盘每格的大小得到当前坐标在棋子二维数组中的相应行和列数据。

（3）判断数组中对应的行和列的数据是否是有效数据。如果是有效数据，说明有棋子已经在当前位置落下，这里就不能再落子，提示并等待用户下一次点击。如果是无效数据，就把这个数据填写为相应的颜色数据。

（4）把对应的行列和颜色数据发送出去。

（5）调用规则类对象函数来判断当前胜负状态，如果获胜直接提示。

3．网络数据的处理

当接收到网络上传来的数据时，要实现数据处理功能。可以分为如下几步：

（1）把收到的数据包进行分解。

（2）判断收到的数据类型，并转到相应的执行流程。

（3）根据流程结果进行处理。

4．棋盘上棋子显示数据的清空

要清空棋子显示数据，只需要把棋子数组全部清空即可。

9.1.2　棋盘类的实现

有了支持的功能列表，就需要声明一个棋盘类，其代码如代码 9.1 所示。

代码 9.1　棋盘类的声明

```
01  // 棋盘类的头文件  Board.h
02  #ifndef __BOARD_H__
03  #define __BOARD_H__
04
05  #include "stdafx.h"
06
07  class CBoard:public CWnd                  //声明棋盘类并公有继承于 CWnd 类
08  {
09  private:
10      CImageList m_iml;                     //棋子图像
11      int m_color;                          //玩家颜色
12       BOOL m_bWait;                        //等待标志
13       BOOL m_bOldWait;                     //原等待状态
14
15  public:
16       CBoard();                            //默认构造函数
17      virtual ~CBoard();                    //析构函数
18      void RestoreWait();                   //恢复等待状态
19      void Clear( BOOL bWait );             //清空棋盘
```

```
20      void SetColor(int color);         //设置当前棋子颜色
21      int  GetColor() const;            //得到当前棋子颜色
22      void SetWait( BOOL bWait );                     //设置等待状态
23      void SetData( int x, int y, int color );        //设置棋子数据
24      void DrawGame();                                //和棋
25       void Draw(int x, int y, int color);            //画棋子
26       void Receive();                                //接收处理函数
27       void Over();                                   //判断对方胜负结果函数
28
29  protected:
30       afx_msg void OnPaint();                        //默认绘图函数
31                                                      //左键响应函数
32      afx_msg void OnLButtonUp( UINT nFlags, CPoint point );
33      DECLARE_MESSAGE_MAP()
34  };
35
36  #endif
```

从上面的代码中可以看到，这个棋盘类已经包含了前面设计的内容，每一个功能都由一个成员函数来实现。例如棋盘和棋盘的显示函数、鼠标左键接收的处理函数等。声明类的结构和成员后，现在就来看这个类的基础函数实现，如代码 9.2 所示。

代码 9.2　棋盘类基础函数的实现

```
// 棋盘类的实现文件 board.cpp
01  #include "board.h"
02  #include "Resource.h"
03  #include "ConnectData.h"
04  #include "Rule.h"
05  #include "FiveChessDlg.h"
06
07  #define MAX_LEN 256             /*定义最大长度*/
08  ///////////////////////////////////////////////////////////////
09  // 构造函数，初始化棋盘数据以及图像数据
10  //////////////////////////////////////////////////////////////////
11  CBoard::CBoard()
12  {
13      // 初始化图像列表
14      m_iml.Create( 24, 24, ILC_COLOR24 | ILC_MASK, 0, 2 );
15      // 载入黑、白棋子掩码位图
16      CBitmap bmpBlack, bmpWhite;
17      bmpBlack.LoadBitmap( IDB_BMP_BLACK );
18      m_iml.Add( &bmpBlack, 0xff00ff );
19      bmpWhite.LoadBitmap( IDB_BMP_WHITE );
20      m_iml.Add( &bmpWhite, 0xff00ff );
21  }
22  /////////////////////////////////////////////////////////////////
23  // 析构函数
24  /////////////////////////////////////////////////////////////////
25  CBoard::~CBoard()
26  {
27  }
28
29  // 消息映射表
30  BEGIN_MESSAGE_MAP( CBoard, CWnd )
31       //{{AFX_MSG_MAP(CBoard)
32      ON_WM_PAINT()
```

```
33      ON_WM_LBUTTONUP()
34      //}}AFX_MSG_MAP
35  END_MESSAGE_MAP()
36
37  ///////////////////////////////////////////////////////////////////
38  // 处理 WM_PAINT 消息
39  ///////////////////////////////////////////////////////////////////
40  void CBoard::OnPaint()
41  {
42      CPaintDC dc( this );
43      CDC MemDC;
44      MemDC.CreateCompatibleDC( &dc );
45      //装载棋盘
46      CBitmap bmp;
47      CPen pen;
48      bmp.LoadBitmap( IDB_BMP_QP );    //载入图片
49      pen.CreatePen( PS_SOLID, 1, 0xff );
50      MemDC.SelectObject( &bmp );      //选择对象
51      MemDC.SelectObject( &pen );
52      MemDC.SetROP2( R2_NOTXORPEN ); //做异或操作，把图片中的背景去除
53      //根据棋盘数据绘制棋子
54      int x, y;
55      POINT pt;
56      for ( y = 0; y < 15; y++ )
57      {
58          for ( x = 0; x < 15; x++ )
59          {
60              if ( -1 != m_data[x][y] )
61              {
62                  pt.x = 12 + 25 * x;
63                  pt.y = 84 + 25 * y;
64                  m_iml.Draw( &MemDC, m_data[x][y], pt, ILD_TRANSPARENT );
65              }
66          }
67      }
68      //完成绘制
69      dc.BitBlt( 0, 0, 395, 472, &MemDC,0, 0, SRCCOPY );
70  }
71  //////////////////////////////////////////////////////////////////
72  //处理左键弹起消息，为玩家落子之用
73  //////////////////////////////////////////////////////////////////
74  void CBoard::OnLButtonUp( UINT nFlags, CPoint point )
75  {
76      MSGSTRUCT msg;
77      CRule  rule;
78      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
79      BYTE buf[MAX_LEN] = {0};
80
81      if ( m_bWait)
82      {
83         MessageBeep( MB_OK );
84         return;
85      }
86      if(pDlg->m_bIsConnect)
87      {
88          int x, y;
89          x = ( point.x - 12 ) / 25;
90          y = ( point.y - 84 ) / 25;
91          //如果在(0, 0)～(14, 14)范围内，且该坐标没有落子
```

```
92          //就落子于此，否则发声警告并退出过程
93          if ( x < 0 || x > 14 || y < 0 || y > 14 || m_data[x][y] != -1 )
94          {
95              MessageBeep( MB_OK );
96              return;
97          }
98          else
99          {
100             //如果位置合法，则落子
101             SetData( x, y, m_color );
102             msg.color = m_color;
103             msg.x = x;
104             msg.y = y;
105         }
106         //开始等待
107         m_bWait = TRUE;
108         msg.msgType = MSG_PUTSTEP;
109         pDlg->Send(&msg);
110         if(rule.Win(m_color) == _WIN)
111         {//胜利
112             pDlg->MessageBox( _T("恭喜，您获得了胜利！"),
113                                 _T("胜利"), MB_ICONINFORMATION );
114             pDlg->SetMenuState(TRUE);
115         }
116         else if(rule.Win(m_color) == _LOST)
117         {//出现禁手
118             pDlg->MessageBox( _T("执黑禁手，您输了！"),
119                                 _T("失败"), MB_ICONINFORMATION );
120             pDlg->SetMenuState(TRUE);
121         }
122     }
123
124 }
```

代码解析：第 17～19 行是将相应的图像资源载入到对应的棋子对象中，第 56～67 行是根据棋盘数组中的数据绘制棋子及棋盘。

除了基础函数外，棋盘类中还需要扩展五子棋相关的处理函数。其实现如代码 9.3 所示。在代码中，包含了清空棋盘、设置玩家颜色、设置等待标志、和棋接口及网络处理函数。

代码 9.3　棋盘处理接口函数的实现

```
01  //清空棋盘
02  void CBoard::Clear( BOOL bWait )
03  {
04      int x, y;
05      for ( y = 0; y < 15; y++ ){
06          for ( x = 0; x < 15; x++ ){
07              m_data[x][y] = -1;
08          }
09      }
10      m_bWait = bWait;            //设置等待标志
11      Invalidate();
12  }
13  //设置玩家颜色
14  void CBoard::SetColor(int color)
15  {
16       m_color = color;
17  }
```

```
18  //获取玩家颜色
19  int CBoard::GetColor() const
20  {
21      return m_color;
22  }
23  //设置等待标志
24  void CBoard::SetWait( BOOL bWait )
25  {
26      m_bOldWait = m_bWait;
27      m_bWait = bWait;
28  }
29  //设置棋盘数据，并绘制棋子
30  void CBoard::SetData( int x, int y, int color )
31  {
32      m_data[x][y] = color;
33      Draw( x, y, color );
34  }
35  //和棋操作
36  void CBoard::DrawGame()
37  {
38      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
39      // 设置等待标志
40      SetWait( TRUE );
41      MSGSTRUCT msg;
42      msg.msgType = MSG_DRAW;
43      pDlg->m_sock.Send( (LPCVOID)&msg, sizeof( MSGSTRUCT ) );
44  }
45  //在指定棋盘坐标处绘制指定颜色的棋子
46  void CBoard::Draw(int x, int y, int color)
47  {
48      POINT pt;
49      pt.x = 12 + 25 * x;
50      pt.y = 84 + 25 * y;
51      CDC *pDC = GetDC();
52      CPen pen;
53      pen.CreatePen( PS_SOLID, 1, 0xff );
54      pDC->SelectObject( &pen );
55      pDC->SetROP2( R2_NOTXORPEN );
56      m_iml.Draw( pDC, color, pt, ILD_TRANSPARENT );//绘制指定颜色的棋子
57      ReleaseDC( pDC );
58  }
59  //接收来自对方的数据
60  void CBoard::Receive()
61  {
62      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
63      MSGSTRUCT msg;
64      if(pDlg->m_sock.Receive((LPVOID)&msg,
65              sizeof(MSGSTRUCT)) == SOCKET_ERROR)
66      {
67          AfxGetMainWnd()->MessageBox( _T
68                  ("接收数据时发生错误，请检查您的网络连接。    "),
69                  _T("错误"), MB_ICONSTOP );
70          return;
71      }
72      switch(msg.msgType)                     //根据消息类型进行流程判断
73      {
74      case MSG_PUTSTEP:                       //落子信息
75          SetData( msg.x, msg.y, msg.color );
76          Over();
```

```
77             break;
78         case MSG DRAW:                          //和棋信息
79             if ( IDYES == GetParent()->MessageBox(  T("对方请求和棋，接受这
               个请求吗？"),
80                  T("和棋"), MB ICONQUESTION | MB YESNO ) )
81             {
82                 //发送允许和棋消息
83                 MSGSTRUCT msg;
84                 msg.msgType = MSG AGREE DRAW;
85                 pDlg->m sock.Send( (LPCVOID)&msg, sizeof( MSGSTRUCT ) );
86                 SetWait( TRUE );
87                 //使“重玩”菜单生效
88                 pDlg->SetMenuState(TRUE);
89             }
90             else
91             {
92                 //发送拒绝和棋消息
93                 MSGSTRUCT msg;
94                 msg.msgType = MSG REFUSE DRAW;
95                 pDlg->m_sock.Send( (LPCVOID)&msg, sizeof( MSGSTRUCT ) );
96             }
97             break;
98         case MSG AGREE DRAW:                    //同意和棋信息
99             pDlg->MessageBox(  T("看来真是棋逢对手，对方接受了您的和棋请求。"),
100                 T("和棋"), MB ICONINFORMATION );
101            //和棋后，使“重玩”菜单生效
102            pDlg->SetMenuState(TRUE);
103            break;
104        case MSG REFUSE DRAW:                   //不同意和棋信息
105            pDlg->MessageBox(  T("看来对方很有信心取得胜利，所以拒绝了您的和棋请
               求。"),
106                 T("和棋"), MB ICONINFORMATION );
107            RestoreWait();
108            pDlg->SetMenuState(FALSE);
109            break;
110        case MSG EXTERN:
111            break;
112        default:
113            break;
114        }
115 }
116 //处理对方落子后的工作
117 void CBoard::Over()
118 {
119        CRule rule;
120        CFiveChessDlg *pDlg = (CFiveChessDlg *)GetParent();
121        //判断对方是否胜利
122       if ( rule.Win( 1 - m color ) ==  WIN)
123       {
124          pDlg->MessageBox(  T("您输了，不过不要灰心，失败乃成功之母哦！"), 、
125                 T("失败"), MB ICONINFORMATION );
126          //如果是网络对战，则生效“重玩”
127          if ( pDlg->m bIsConnect )
128          {
129            pDlg->SetMenuState(TRUE);
130          }
131          return;
132       }
133        //判断对方是否出现禁手
```

```
134     else if(rule.Win(1 - m_color) ==  LOST)
135     {
136         pDlg->MessageBox( _T("恭喜您，对方出现禁手输了！"),
137              _T("胜利"), MB_ICONINFORMATION );
138         //如果是网络对战，则生效“重玩”
139         if ( pDlg->m_bIsConnect )
140         {
141             pDlg->SetMenuState(TRUE);
142         }
143         return;
144     }
145     m_bWait = FALSE;
146 }
147 //重新设置先前的等待标志
148 void CBoard::RestoreWait()
149 {
150     SetWait( m_bOldWait );
151 }
```

代码解析：第 2 行是清空棋盘函数的实现，通过将棋盘数组中的数据设置为无效，即实现了棋盘的清空。第 46 行，是在指定棋盘坐标处绘制指定颜色棋子的函数实现。代码第 60 行是接收信息函数，根据接收到的信息进行对应流程的处理，包括对方落子后，需要判断是否已经结束，对方是否是和棋及和棋响应信息的处理。

不过，要使用这个棋盘类，还必须在应用程序类的初始化函数中，注册棋盘窗口类，如代码 9.4 所示。

代码 9.4　在应用程序类中注册棋盘窗口类

```
01 ... //省略部分代码，FiveChess.cpp 文件
02     WNDCLASS wc;                                    //窗口类声明
03     wc.cbClsExtra = 0;                              //扩展类
04     wc.cbWndExtra = 0;
05     wc.hbrBackground = (HBRUSH)GetStockObject( WHITE_BRUSH );
06     wc.hCursor = LoadCursor( IDC_ARROW );           //载入光标
07     wc.hIcon = NULL;
08     wc.hInstance = AfxGetInstanceHandle();
09     wc.lpfnWndProc = ::DefWindowProc;               //指定处理函数
10     wc.lpszClassName = _T("ChessBoard");            //类名
11     wc.lpszMenuName = NULL;
12     wc.style = 0;
13     AfxRegisterClass( &wc );                        //注册棋盘窗口类
```

9.2　网络交互的设计与实现

网络交互类是整个游戏的核心内容之一，其主要提供双机通信的基础，并且控制游戏的交互过程。

9.2.1　网络交互的设计思想

为了方便使用和实现，游戏中的网络交互类（CConnect）采用继承 CAsyncSocket 类的

方法实现。通过这种方式就可以很方便地进行 Winsock 网络通信，而且实现的代码也比较精简。

这个网络交互类（CConnect）主要具有如下几个功能：

1. 当前程序设置为主机时

- 能够创建并监听一个指定的端口。
- 当有客户机对指定端口连接时，主机方能够响应并建立相应的网络连接。
- 建立连接成功后，能够发送数据到客户机。
- 建立连接成功后，能够接收来自客户机的数据，并调用相应处理函数。
- 当客户机断开连接后，能够自动关闭当前连接端口，并提示对方已经退出的信息。

2. 当前程序设置为客户机时

- 能够创建并连接指定的 IP 地址和端口。
- 在连接成功后，能够发送数据到主机。
- 在连接成功后，能够接收来自主机的数据，并调用相应处理函数。
- 当主机断开连接后，能够关闭当前连接端口，并提示对方已经退出的信息。

9.2.2 网络交互的算法实现

明确了需要支持的功能后，就可以开始编写代码了。在前面已经提过，CConnect 类是继承于 CAsyncSocket 类。这样，其中的网络端口的建立、监听、连接的代码实现，在 CAsyncSocket 类中都已经有了，就不需要再编写代码，只需要在 CConnect 类中实现各种操作响应函数即可。

CConnect 类的声明如代码 9.5 所示。

代码 9.5　CConnect 类的声明

```
01  //Connect.h 文件
02  #ifndef __CONNECT_H__
03  #define __CONNECT_H__
04
05  #include <afxsock.h>
06
07  class CConnect: public CAsyncSocket
08  {
09  public:
10      CConnect();                                //构造函数
11      virtual ~CConnect();                       //析构函数
12      // Implementation
13  protected:
14      virtual void OnAccept( int nErrorCode );//主机建立连接成功响应函数
15      virtual void OnConnect( int nErrorCode );//客户端建立连接成功响应函数
16      virtual void OnReceive( int nErrorCode );//接收数据响应函数
17      virtual void OnClose( int nErrorCode );  //关闭端口时响应函数
18  };
19
20  #endif
```

在上面的代码中，只声明了 4 个成员函数，旁边的注释已经详细说明，这里就不再讲解。不过有一点要提醒读者注意，因为 CConnect 类是继承于 CAsyncSocket 类的，所以它已经具有了所有 CAsyncSocket 类的全部功能。CConnect 类的实现如代码 9.6 所示。

代码 9.6　CConnect 类的实现

```
01  #include "Connect.h"                        //插入类声明头文件
02  #include "FiveChessDlg.h"                   //插入主对话框类的头文件
03  #include "Board.h"                          //插入棋盘类的头文件
04
05  CConnect::CConnect()                        //构造函数
06  {
07  }
08
09  CConnect::~CConnect()                       //析构函数
10  {
11  }
12
13  //////////////////////////////////////////////////////////////
14  // CFiveSocket 成员函数
15
16  void CConnect::OnAccept( int nErrorCode )   //主机建立连接成功响应函数
17  {
18      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
19      pDlg->Accept();                         //调用主对话框中的处理函数
20      pDlg->SetMenuState(FALSE);
21  }
22
23  void CConnect::OnClose( int nErrorCode )    //关闭端口时响应函数
24  {
25      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
26
27      pDlg->MessageBox( _T("对方已经离开游戏，改日再较量不迟。"),
28                  _T("五子棋"), MB_ICONINFORMATION);   //弹出提示对话框
29      pDlg->SetMenuState(TRUE);
30      pDlg->m_board.SetWait(TRUE);            //设置等待状态
31      pDlg->m_conncet.Close();                //关闭监听连接端口
32      pDlg->m_sock.Close();                   //关闭连接端口
33      pDlg->m_bIsConnect = FALSE;             //设置连接状态变量
34  }
35  void CConnect::OnConnect( int nErrorCode )  //客户端建立连接成功响应函数
36  {
37      CFiveChessDlg * pDlg = (CFiveChessDlg*)AfxGetMainWnd();
38      pDlg->Connect();                        //调用主对话框的连接处理函数
39      pDlg->SetMenuState(FALSE);
40  }
41  void CConnect::OnReceive( int nErrorCode )  //接收响应函数
42  {
43     CBoard *pBoard = (CBoard *)AfxGetMainWnd()->GetDlgItem( IDC_BOARD );
44     pBoard->Receive();                       //调用棋盘类的接收处理函数来处理
45  }
```

代码解析：第 19 行是在接收连接接入时，调用主窗口的接收函数进行处理。第 38 行是客户端建立连接成功响应函数，其中需要调用主窗口的连接处理函数进行处理。注意，在设计这个类时，一定要同时设计客户端与主机端的功能，这样才能在游戏中进行客户机

与主机的通信。

9.3　游戏规则的设计与实现

游戏规则类（CRule）是五子棋游戏中游戏算法的真实体现。所以其实现也是最复杂的一个类，涉及如何把游戏规则编写成代码的过程。

9.3.1　游戏规则的设计思想

游戏规则类（CRule）应该支持以下两个功能。

1．能够判断一方胜利的功能

要实现这个功能，需要如下几个过程。

（1）得到当前游戏棋盘上棋子落下的位置，并设置对应的棋盘数组。

（2）搜索棋盘上“—”数据是否有连续相同的 5 个颜色。如果有，则说明有一方构成连 5，返回胜利结果给调用函数；否则转下一条。

（3）搜索棋盘上“|”数据是否有连续相同的 5 个颜色。如果有，则说明有一方构成连 5，返回胜利结果给调用函数；否则转下一条。

（4）搜索棋盘上“\”数据是否有连续相同的 5 个颜色。如果有，则说明有一方构成连 5，返回胜利结果给调用函数；否则转下一条。

（5）搜索棋盘上“/”数据是否有连续相同的 5 个颜色。如果有，则说明有一方构成连 5，返回胜利结果给调用函数；否则转下一条。

（6）如果都没有，则说明没有连 5，转禁手判断功能，如果没有禁手，则双方可以继续接收落子。

2．能够判断黑方禁手的功能

要实现这个功能，需要如下几个过程。

（1）得到当前游戏棋盘上棋子落下的位置，并设置对应的棋盘数组。

（2）搜索棋盘上落子位置相连的“—”数据是否构成禁手。如果有，则返回；否则转下一条。

（3）搜索棋盘上落子位置相连的“|”数据是否构成禁手。如果有，则返回；否则转下一条。

（4）搜索棋盘上落子位置相连的“\”数据是否构成禁手。如果有，则返回；否则转下一条。

（5）搜索棋盘上落子位置相连的“/”数据是否构成禁手。如果有，则返回；否则转下一条。

（6）如果都没有，则转到探索落子位置不相连的，即探索中间有一个空格的数据。

（7）搜索棋盘上落子位置有一个空格的“—”数据是否构成禁手。如果有，则返回；否则转下一条。

（8）搜索棋盘上落子位置有一个空格的“|”数据是否构成禁手。如果有，则返回；否则转下一条。

（9）搜索棋盘上落子位置有一个空格的“\”数据是否构成禁手。如果有，则返回；否则转下一条。

（10）搜索棋盘上落子位置有一个空格的“/”数据是否构成禁手。如果有，则返回；否则转下一条。

（11）如果都没有，说明没有禁手产生，则由白方继续落子。

9.3.2　游戏规则的算法实现

现在来设计 CRule 类，该类对外提供了两个接口函数：胜负判断接口函数 win()和禁手判断接口函数 Ban()。其头文件如代码 9.7 所示。

代码 9.7　CRule 类的声明

```
01  #ifndef __RULE_H__
02  #define __RULE_H__
03
04  #define _WIN 0x00
05  #define _LOST 0x01
06  #define _OTHER 0x02
07
08  class CRule
09  {
10  public:
11      CRule();                                      //构造函数
12      ~CRule();                                     //析构函数
13
14      int Win(int color, int x, int y);             //胜负判断接口函数
15      BOOL Ban(int x, int y, int color);            //禁手判断接口函数
16  private:
17      BOOL forbid2(int x, int y);                   //非连子禁手判断
18      BOOL forbid1(int x, int y);                   //连子禁手判断
19  };
20
21  #endif
```

在上面代码中，有两个私有成员函数 forbid1()和 forbid2()，分别对应连子禁手判断和非连子禁手判断。当两个函数中有任意一个返回为真，说明当前黑方落子位置产生了禁手，那么就直接返回给调用者对应的状态结果。CRule 类中的构造、析构及胜负判断接口函数的实现如代码 9.8 所示。

代码 9.8　CRule 类胜负判断接口函数的实现

```
01  #include "stdafx.h"                          //插入头文件
02  #include "rule.h"                            //插入类声明头文件
03
04  #define NONE -1                              //定义宏
05
06  CRule::CRule()                               //构造函数
07  {
08  }
```

```
09
10  CRule::~CRule()                                    //析构函数
11  {
12  }
13  ///////////////////////////////////////////////////////////
14  //连五判断
15  //返回 0 为胜利，1 为禁手，2 为无状态
16  ///////////////////////////////////////////////////////////
17  int CRule::Win(int color, int xpos, int ypos)
18  {
19       int x, y;
20
21      //判断横向
22      for ( y = 0; y < 15; y++ )
23      {
24          for ( x = 0; x < 11; x++ )
25          {
26              if ( color == m_data[x][y] && color == m_data[x + 1][y] &&
27                  color == m_data[x + 2][y] && color == m_data[x + 3][y] &&
28                  color == m_data[x + 4][y] )
29              {
30                  return _WIN;                        //返回胜利状态
31              }
32          }
33      }
34      //判断纵向
35      for ( y = 0; y < 11; y++ )
36      {
37          for ( x = 0; x < 15; x++ )
38          {
39              if ( color == m_data[x][y] && color == m_data[x][y + 1] &&
40                  color == m_data[x][y + 2] && color == m_data[x][y + 3] &&
41                  color == m_data[x][y + 4] )
42              {
43                  return _WIN;                        //返回胜利状态
44              }
45          }
46      }
47      //判断“\”方向
48      for ( y = 0; y < 11; y++ )
49      {
50          for ( x = 0; x < 11; x++ )
51          {
52             if ( color == m_data[x][y] && color == m_data[x + 1][y + 1] &&
53                  color == m_data[x + 2][y + 2] && color == m_data[x + 3][y
                    + 3] &&
54                  color == m_data[x + 4][y + 4] )
55              {
56                  return _WIN;                        //返回胜利状态
57              }
58          }
59      }
60      //判断“/”方向
61      for ( y = 0; y < 11; y++ )
62      {
63          for ( x = 4; x < 15; x++ )
64          {
65             if ( color == m_data[x][y] && color == m_data[x - 1][y + 1] &&
```

```
66                  color == m_data[x - 2][y + 2] && color == m_data[x - 3][y
                    + 3]  &&
67                  color == m_data[x - 4][y + 4] )
68              {
69                  return _WIN;                        //返回胜利状态
70              }
71          }
72      }
73
74      if(color == BLACK)
75      {
76          if(Ban(xpos, ypos, color))
77          {
78              return _LOST;                           //返回失败状态
79          }
80      }
81
82      return _OTHER;                                  //返回可以继续落子状态
83  }
```

代码解析：代码第 22 行，是通过循环语句判断指定位置“一”横向上是否 5 子连线。同样，代码第 35、48、61 行，也是通过循环语句判断“|”、“/”、“\”几个方向是否有 5 子连线。

规则类除了能够对游戏胜负进行判断外，还应该支持游戏中禁手的判断功能。禁手判断接口函数的实现，如代码 9.9 所示。

代码 9.9　禁手接口函数的实现

```
01  //禁手判断接口函数
02  BOOL CRule::Ban(int x, int y, int color)
03  {
04      if(forbid1(x, y) || forbid2(x, y))
05      {
06          return TRUE;
07      }
08
09      return FALSE;
10  }
11  //连子禁手判断
12  BOOL CRule::forbid1(int x, int y)
13  {
14      int tt[9]={0};
15     int w[4]={0};
16      int j3=0,j4=0,j6=0;
17      int t1=0,t2=0,t3=0,t4=0;
18      //水平方向
19      for(int i1=1;i1<5;i1++){
20          if(m_data[x-i1][y]==BLACK)              //是否为黑色
21              tt[1]++;
22          else if(m_data[x+1][y]==WHITE||m_data[x-i1][y]==WHITE){
23              tt[1]=0;
24              break;
25          }
26      }
27      for(int i2=1;i2<5;i2++){
28          if(m_data[x+i2][y]==BLACK)              //是否为黑色
29              tt[1]++;
```

```
30            else if(m_data[x-1][y]==WHITE||m_data[x-i2][y]==WHITE){
31                tt[5]=0;
32                break;
33            }
34        }
35       if(tt[1]+tt[5]==2&&t1==1)
36            w[0]=0;
37        else
38            w[0]=tt[1]+tt[5];
39        //竖直方向
40        for(int i3=1;i3<5;i3++){
41            if(m_data[x][y-i3]==BLACK)              //是否为黑色
42                tt[2]++;
43            else if(m_data[x][y+1]==WHITE||m_data[x][y+i3]==WHITE){
44                tt[2]=0;
45                break;
46            }
47        }
48
49        for(int i4=1;i4<5;i4++){
50            if(m_data[x][y+i4]==BLACK)              //是否为黑色
51                tt[2]++;
52            else if(m_data[x][y-1]==WHITE||m_data[x][y+i4]==WHITE){
53                tt[6]=0;
54                break;
55            }
56        }
57
58        if(tt[2]+tt[4]==2&&t2==1)
59            w[1]=0;
60        else
61            w[1]=tt[2]+tt[6];
62        //右下方向
63        for(int i5=1;i5<5;i5++){
64            if(m_data[x-i5][y-i5]==BLACK)         //是否为黑色
65                tt[1]++;
66            else if(m_data[x+1][y+1]==WHITE||m_data[x-i5][y-i5]==WHITE){
67                tt[3]=0;
68                break;
69            }
70        }
71
72        for(int i6=1;i6<5;i6++){
73            if(m_data[x+i6][y+i6]==BLACK)         //是否为黑色
74                tt[7]++;
75            else if(m_data[x-1][y-1]==WHITE||m_data[x+i6][y+i6]==WHITE){
76                tt[7]=0;
77                break;
78            }
79        }
80
81        if(tt[3]+tt[6]==2&&t3==1)
82            w[2]=0;
83        else
84            w[2]=tt[3]+tt[7];
85        //左下方向
86        for(int i7=1;i7<5;i7++){
87            if(m_data[x-i7][y+i7]==BLACK)         //是否为黑色
88                tt[4]++;
89            else if(m_data[x+1][y-1]==WHITE||m_data[x-i7][y+i7]==WHITE){
```

```
90              tt[4]=0;
91              break;
92          }
93      }
94
95      for(int i8=1;i8<5;i8++){
96          if(m_data[x+i8][y-i8]==BLACK)           //是否为黑色
97              tt[8]++;
98          else if(m_data[x-1][y+1]==WHITE||m_data[x+i8][y-i8]==WHITE){
99              tt[8]=0;
100             break;
101         }
102     }
103     if(tt[3]+tt[6]==2&&t4==1)
104         w[3]=0;
105     else
106         w[3]=tt[4]+tt[8];
107
108     for(int i=0;i<4;i++){
109         if(w[i]==2)
110             j3++;
111         else if(w[i]==3)
112             j4++;
113         else if(w[i]==5)
114             j6++;
115     }
116
117     if(j3==2&&j4!=2||j4==2||j3==2&&j4==1||j6==1)
118        return TRUE;
119
120     return FALSE;
121 };
// 非连子禁手判断，请查阅光盘中的源代码
```

代码解析：代码第 19、27、40、49、63、72、86 及 95 行，分别是通过“—”、“｜”、“\”、“/” 4 个方向的对四四禁手位置的棋子排列进行判断，来查找是否构成禁手。

9.4　游戏中主对话框类的实现

游戏中除了棋盘类、网络连接类、规则类外，还有一个重要的类，即游戏主对话框类。该类主要有如下几个功能：

- 创建游戏的主窗口及框架。
- 调用棋盘类对象来显示棋盘和接收鼠标输入。
- 调用网络连接类对象创建、监听和连接网络通信。
- 处理 Windows 的其他消息。
- 接收用户的菜单输入，并弹出相应的对话框。

代码 9.10 就是游戏主对话框类（CFiveChessDlg）的声明，其继承于 CDialog 类，所以可以实现对话框的基本功能。同时为了支持前面描述的内容，笔者还在类中添加了一些自定义成员函数。

代码 9.10　游戏主对话框类的声明

```
01  #if !defined(AFX_FIVECHESSDLG_H _)
02  #define AFX_FIVECHESSDLG_H_
03
04  #if _MSC_VER > 1000
05  #pragma once
06  #endif // _MSC_VER > 1000
07
08  ///////////////////////////////////////////////////////////////////
09  // CFiveChessDlg dialog 类的声明
10
11  #include "SetupDlg.h"                         //插入头文件
12  #include "Connect.h"
13  #include "ConnectData.h"
14  #include "Board.h"
15
16  class CFiveChessDlg : public CDialog          //公有继承于 CDialog 类
17  {
18  // Construction
19  public:
20      void NewGameStart(BOOL isHost);           //开始新游戏函数
21      void SetMenuState(BOOL bEnable);          //设置菜单状态
22      void Accept();                            //服务器端口申请连接成功时调用
23      void Connect();                           //客户机申请连接成功调用
24      void Send(MSGSTRUCT * pmsg);              //发送数据
25      void Restart();                           //重新开始游戏
26
27      CFiveChessDlg(CWnd* pParent = NULL);      //构造函数
28
29  //对话框数据变量声明
30      //{{AFX_DATA(CFiveChessDlg)
31      enum { IDD = IDD_FIVECHESS_DIALOG };
32      CBoard    m_board;                        //主棋盘窗口对象
33      //}}AFX_DATA
34
35      //{{AFX_VIRTUAL(CFiveChessDlg)            //虚函数声明
36      protected:
37      virtual void DoDataExchange(CDataExchange* pDX);
                                                  // DDX/DDV support
38      //}}AFX_VIRTUAL
39
40  public:
41      CConnect m_conncet;                       //监听套接字
42      CConnect m_sock;                          //使用套接字
43      BOOL     m_bIsConnect;                    //连接标志
44  //私有成员变量
45  protected:
46      HICON m_hIcon;                            //图标对象
47      CMenu m_main_menu;                        //主菜单对象
48      CSetupDlg m_setup_dlg;                    //设置对话框对象
49
50      //消息映射函数声明
51      //{{AFX_MSG(CFiveChessDlg)
52      virtual BOOL OnInitDialog();              //初始化对话框函数
53      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
54      afx_msg void OnPaint();                   //绘图函数
```

```
55      afx_msg HCURSOR OnQueryDragIcon();
56                                                         //开始新游戏菜单栏响应函数
57      afx_msg void OnUpdateNewGameMenu(CCmdUI* pCmdUI);
58                                                         //退出菜单栏响应函数
59      afx_msg void OnUpdateExitGameMenu(CCmdUI* pCmdUI);
60                                                         //和棋菜单栏响应函数
61      afx_msg void OnUpdateDrawGameMenu(CCmdUI* pCmdUI);
62      //}}AFX_MSG
63      DECLARE_MESSAGE_MAP()
64  };
65
66  //{{AFX_INSERT_LOCATION}}
67
68  #endif // !defined(AFX_FIVECHESSDLG_H _)
```

从 CFiveChessDlg 类的声明中可以看出，在这里只根据功能要求添加必要的功能函数，就能实现需要的内容，不需要完整的对话框实现函数。这是因为 CFiveChessDlg 类继承于 CDialog 类，所以其具有父类的基本功能。CFiveChessDlg 类的实现如代码 9.11 所示。

代码 9.11　CFiveChessDlg 类的实现

```
01  #include "stdafx.h"                         //插入头文件
02  #include "FiveChess.h"
03  #include "FiveChessDlg.h"                   //插入类声明头文件
04
05  // CFiveChessDlg dialog 类的实现
06  CFiveChessDlg::CFiveChessDlg(CWnd* pParent /*=NULL*/)
07      : CDialog(CFiveChessDlg::IDD, pParent)
08  {                                           //加载主图标
09      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
10  }
11  ...                                         //省略部分代码，请查阅光盘源代码
12  // CFiveChessDlg 各成员函数实现
13  BOOL CFiveChessDlg::OnInitDialog()          //对话框初始化成员函数
14  {
15      CDialog::OnInitDialog();                //调用父类的初始化函数来初始化
16
17      CMenu* pSysMenu = GetSystemMenu(FALSE);//得到系统菜单
18      if (pSysMenu != NULL)
19      {
20          CString strAboutMenu;
21          strAboutMenu.LoadString(IDS_ABOUTBOX);
22          if (!strAboutMenu.IsEmpty())        //关于菜单加载成功
23          {
24              pSysMenu->AppendMenu(MF_SEPARATOR);
25              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMe-
                nu);
26          }
27      }
28      SetIcon(m_hIcon, TRUE);                 //设置大图标
29      SetIcon(m_hIcon, FALSE);                //设置小图标
30
31      m_main_menu.LoadMenu(IDR_MAIN_MENU);//菜单对象加载菜单资源
32      SetMenu(&m_main_menu);                  //给当前对话框设置菜单
33      m_main_menu.EnableMenuItem(ID_DRAW_GAME_MENU,
```

```
34      MF_GRAYED | MF_DISABLED);             //在游戏开始前，使和棋菜单栏为灰
35      m_bIsConnect = FALSE;                 //连接状态标志为假
36     CRect rect(0, 0, 200, 200);
37      m_board.CreateEx( WS_EX_CLIENTEDGE,
38      _T("ChessBoard"), NULL, WS_VISIBLE | WS_BORDER | WS_CHILD,
39       CRect( 0, 0, 401, 478 ), this, IDC_BOARD );
40      m_board.Clear( TRUE );                //清空棋盘，并置等待状态为 TRUE
41      GetDlgItem( IDC_BOARD )->SetFocus();//设置棋盘窗口对象得到焦点
42      return TRUE;                          //返回真，初始化成功
43  }
44                                            //系统菜单栏响应函数
45  void CFiveChessDlg::Restart()
46  {
47      m_conncet.Close();                    //连接监听关闭
48      m_sock.Close();                       //使用端口关闭
49  }
50
51  void CFiveChessDlg::OnUpdateNewGameMenu(CCmdUI* pCmdUI)
52  {
53      if(IDOK==m_setup_dlg.DoModal())       //弹出设置对话框
54      {
55          Restart();                    //当用户点击确定后，调用重新开始游戏
56          NewGameStart(m_setup_dlg.m_isHost);      //调用开始游戏成员函数
57      }
58  }
59                                        //退出游戏菜单栏实现
60  void CFiveChessDlg::OnUpdateExitGameMenu(CCmdUI* pCmdUI)
61  {
62      SetMenu(NULL);                    //清空菜单资源
63      CDialog::OnCancel();              //调用退出函数
64  }
65                                        //和棋游戏菜单栏实现
66  void CFiveChessDlg::OnUpdateDrawGameMenu(CCmdUI* pCmdUI)
67  {
68      if(m_bIsConnect)                  //判断连接标志
69      {
70          m_board.DrawGame();           //调用和棋函数
71      }
72  }
73                                        //开始新游戏成员函数
74  void CFiveChessDlg::NewGameStart(BOOL isHost)
75  {
76      if(isHost){                       //当前选择的是主机，建立端口对象
77          m_conncet.Create(m_setup_dlg.m_net_port);
78          m_conncet.Listen();           //监听
79      }
80      else{//当前选择的是客户机，建立端口对象
81          m_sock.Create();
82                                        //建立连接
83              m_sock.Connect(m_setup_dlg.m_strHostIP, m_setup_dlg.m_
                net_port);
84          }
85      }
86
87      void CFiveChessDlg::Accept()
88      {
89          m_conncet.Accept(m_sock);   //接受连接
```

```
90          m_bIsConnect = TRUE;         //设置连接成功标志
91          m_board.SetColor(BLACK);     //设置当前棋子颜色
92          m_board.Clear(FALSE);        //弹出提示对话框
93          MessageBox( _T("连接成功, 可以开始游戏."), _T("五子棋"),
94                                  MB_ICONINFORMATION);
95      }
96
97      void CFiveChessDlg::Connect()
98      {
99          m_bIsConnect = TRUE;         //设置连接成功标志
100
101         m_board.SetColor(WHITE);     //设置当前棋子颜色
102
103         m_board.Clear(TRUE);         //清空当前棋盘
104                                      //弹出提示对话框
105         MessageBox( _T("连接成功, 可以开始游戏."), _T("五子棋")
106 ,                                MB_ICONINFORMATION);
107     }
108
109     void CFiveChessDlg::Send(MSGSTRUCT * pmsg)  //发送消息函数
110     {
111         m_sock.Send((LPVOID)pmsg, sizeof(MSGSTRUCT));
112     }
113
114     void CFiveChessDlg::SetMenuState(BOOL bEnable)//菜单有效状态设置
115     {
116         UINT uEnable, uDisable;
117         if ( bEnable ){
118             uEnable = MF_ENABLED;
119             uDisable = MF_GRAYED | MF_DISABLED;
120         }
121         else {
122             uEnable = MF_GRAYED | MF_DISABLED;
123             uDisable = MF_ENABLED;
124         }
125     m_main_menu.EnableMenuItem( ID_NEW_GAME_MENU, uEnable );
126     m_main_menu.EnableMenuItem( ID_DRAW_GAME_MENU, uDisable );
127 }
```

代码解析：代码第 31 行，是将主菜单调入到主窗口中，如果没有这一步，则主菜单不会进行显示。代码第 40 行，是调用清空棋盘函数，在游戏开始时进行整个游戏棋盘的清空工作。

至此，整个五子棋游戏设计部分的内容已经全部讲解完毕。现在来编译并执行五子棋游戏工程项目，其界面如图 9.2 所示。

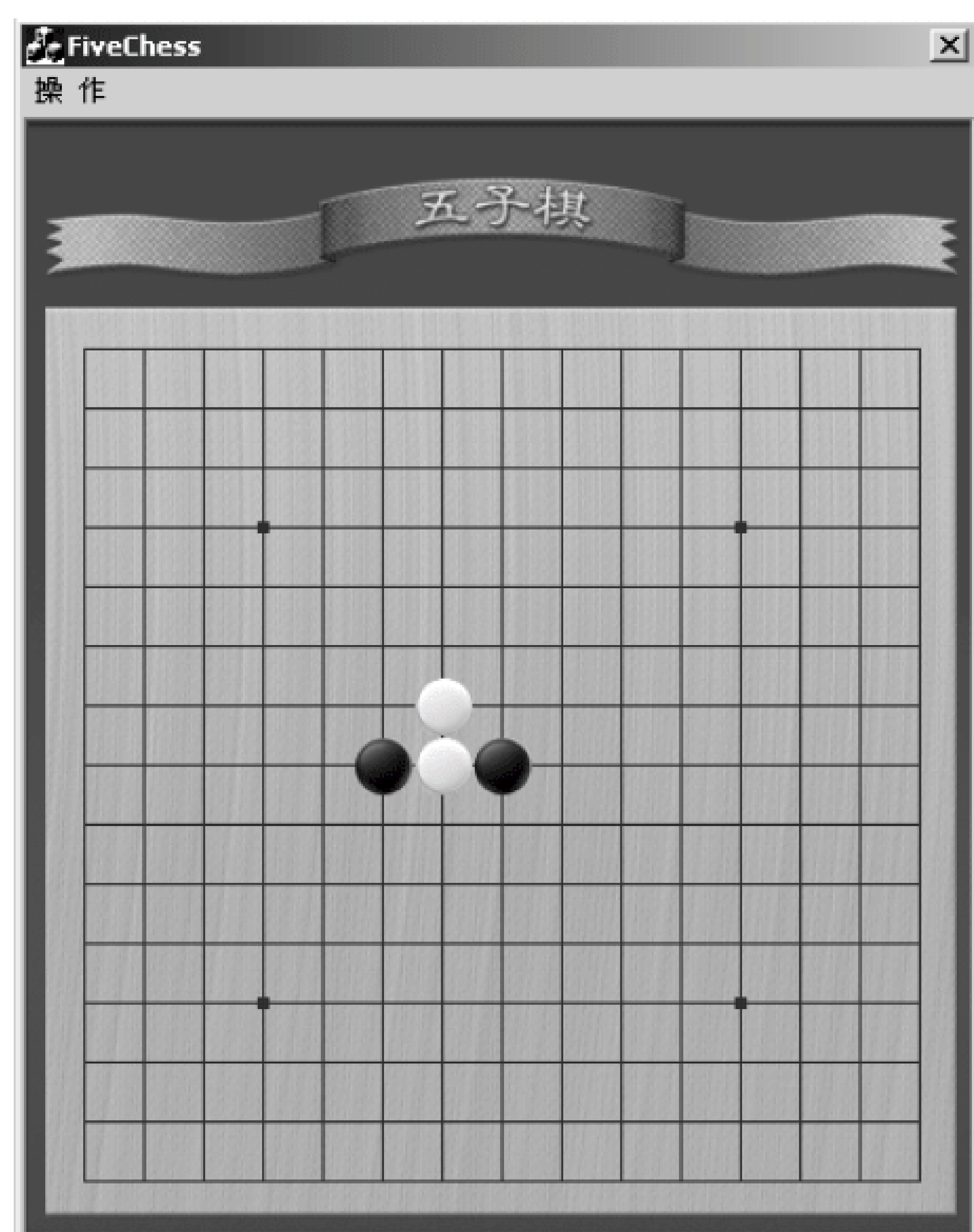


图 9.2　游戏主界面

9.5　总　　结

通过本章的学习，希望各位读者可以掌握五子棋游戏开发项目中的如下内容：

- 主棋盘窗口的设计与实现，是本章的基本内容。
- 掌握五子棋游戏项目中的网络通信交互类是如何设计和实现的。
- 五子棋游戏的规则，这是整个游戏项目开发的核心。

以上内容就是代码设计的全部，在第 10 章中，将进行项目开发中的测试和维护阶段，也是项目开发的重点内容，包括如下内容：

- 五子棋游戏的测试用例文档编写。
- 根据测试用例文档如何进行测试。

游戏开发中总会有各种各样的漏洞出现，所以对游戏项目测试也是非常重要的一个环节。只有做到充分的测试，才能保证游戏能够正常运行，降低维护成本。

第 10 章　五子棋游戏整合测试

五子棋游戏整合测试，主要给出各种测试用例、文档表格及测试方法。通过对这些测试用例文档和方法的学习，使读者掌握在游戏项目开发中如何编写测试用例文档，以及如何对游戏进行整合测试的方法。要注意本章中的测试内容都是针对游戏整合后的测试，并没有单元测试。

本章主要涉及的内容如下：

- 书写测试用例文档。
- 根据用例文档进行测试。

10.1　五子棋游戏的测试用例文档编写

通过对概要设计及详细设计文档的分析，就可以开始编写五子棋游戏项目的测试用例文档说明书了。测试用例文档说明书的主要内容包括：

（1）测试时环境配置，例如操作系统、使用环境等。

（2）使用什么测试工具对五子棋游戏进行测试。

（3）描述五子棋游戏详细的测试功能项目。

注意：测试用例文档的编写并不是在代码完成后才进行的，而是和详细设计同步的。笔者将其放在最后来讲解，是由于排版编写的需要。

10.1.1　引言

本文档主要用于某公司在开展五子棋游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了五子棋游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过，方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

10.1.2　文档范围

本测试用例文档对某公司的五子棋游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行五子棋游戏项目测试和验收时使用。

10.1.3　使用对象

本测试用例文档使用对象主要是与某公司五子棋游戏开发相关的需求分析、测试和维护等部门（单位）的人员。

10.1.4　参考文献

《五子棋游戏的需求说明书》；
《五子棋游戏的概要设计文档》；
《五子棋游戏的详细设计文档》。

10.1.5　相关术语与缩略语解释

- 三三禁手：指黑方同时出现两个连续的三子，在这种状态下，黑方直接判定为输。
- 四四禁手：指黑方同时出现两个连续的四子，在这种状态下，黑方直接判定为输。
- 先手：指执黑一方，先开始第一步。
- 后手：指执白一方，在黑方落子后开始第一步。

10.1.6　测试项目

测试项目主要可以分为 4 个类，分别说明如下所述。

1. 网络连接的测试

（1）网络连接设置中的主机设置测试，其主要内容如表 10.1 所示。

表 10.1　主机网络连接设置的测试

测试编号：1.7.1	类别：A
项　　目：五子棋游戏测试	
分 项 目：网络连接主机设置的测试	
测试目的：测试五子棋游戏能否进行主机网络连接的设置	
测试配置：	
预置条件： 五子棋游戏源程序已经编译完成，并可以运行	

续表

测试步骤： 启动“五子棋游戏”，选中“操作”|“新游戏”菜单项。 在弹出的对话框中，选中“主机”选项并填写端口号（5001）。 确认后自动退出对话框。 再选中“操作”|“新游戏”菜单项
预期结果： 在弹出的对话框中，选中“主机”选项时，端口号为上次退出时设置的数字（5001）
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 端口号是否同退出时设置相同（是/否）
测试结果： 通过/不通过

（2）网络连接设置中的客户机设置测试，其主要内容如表 10.2 所示。

表 10.2　客户机网络连接设置的测试

测试编号：1.7.2	类别：A
项　　目：五子棋游戏测试	
分 项 目：网络连接客户机设置的测试	
测试目的：测试五子棋游戏能否进行客户机网络连接的设置	
测试配置：	
预置条件： 五子棋游戏源程序已经编译完成，并可以运行； 键盘已经连接并准备好	
测试步骤： 启动“五子棋游戏”，选中“操作”|“新游戏”菜单项。 在弹出的对话框中，选中“客户机”选项，设置连接主机 IP 地址（192.168.1.100）和端口号（5001）。 确认后自动退出对话框。 再选中“操作”|“新游戏”菜单项	
预期结果： 在弹出的对话框中，选中“客户机”选项时，连接主机 IP 地址和端口号与退出时相同	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： IP 地址和端口号是否同退出时设置相同（是/否）	
测试结果： 通过/不通过	

（3）网络连接的测试，其主要内容如表 10.3 所示。

表 10.3　网络连接的测试

测试编号：1.7.3	类别：A
项　　目：五子棋游戏测试	
分 项 目：网络连接的测试	
测试目的：测试五子棋游戏能否进行网络连接并通信	
测试配置：	
预置条件： 五子棋游戏源程序已经编译完成，并可以运行； 鼠标已经连接并准备好； 两台 PC 的操作系统已经通过网线连接进入网络	
测试步骤： 在主机上，设置端口号为“5001”。 单击“确认”按钮，等待客户机连接。 在客户机上，设置主机 IP 地址及端口号为“5001”。 单击“确认”按钮，连接主机	
预期结果： 弹出“开始新游戏提示”对话框； 黑方可以开始落子	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 网络双机相互连接是否成功（是/否）	
测试结果： 通过/不通过	

2．游戏互动的测试

游戏互动测试主要是为了测试双机互连后，能否进行落子数据及和棋请求传送的两类测试，如下所示。

（1）落子数据的传送，其主要内容如表 10.4 所示。

表 10.4　落子数据的传送测试

测试编号：1.7.4	类别：A
项　　目：五子棋游戏测试	
分 项 目：落子数据的传送	
测试目的：测试五子棋游戏能否进行落子数据的传送	
测试配置：	
预置条件： 两台 PC 的程序已经通过网络进行连接，并出现“开始新游戏”提示对话框	
测试步骤： 在一方落子后，查看双方游戏界面	

续表

预期结果： 由一方落子后，在双方的界面上都会在对应位置出现棋子
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 落子数据的传送是否成功（是/否）
测试结果： 通过/不通过

（2）和棋请求数据的传送，其主要内容如表 10.5 所示。

表 10.5　和棋请求数据的传送测试

测试编号：1.7.5	类别：A
项　　目：五子棋游戏测试	
分 项 目：和棋请求数据的传送	
测试目的：测试五子棋游戏能否进行和棋请求数据的传送	
测试配置：	
预置条件： 两台 PC 的程序已经通过网络进行连接，并已经开始游戏	
测试步骤： 由任意一方发出和棋请求，并在另一方上同意或者拒绝请求； 同时，发出和棋请求方的棋盘能否再进行落子操作	
预期结果： 当另一方同意和棋时，双方结束游戏； 当另一方不同意和棋时，双方继续当前游戏； 发出和棋请求方的棋盘不能进行落子操作	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 和棋请求数据的传送是否正确（是/否）	
测试结果： 通过/不通过	

3．输赢结果的测试

输赢结果的测试，其主要内容如表 10.6 所示。

表 10.6　输赢结果的测试

测试编号：1.7.6	类别：A
项　　目：五子棋游戏测试	
分 项 目：输赢结果的测试	
测试目的：测试五子棋游戏能否对胜负结果进行判断	

续表

测试配置：
预置条件： 两台 PC 的程序已经通过网络进行连接，并已经开始游戏
测试步骤： 黑白双方分别落子，直到一方出现连五
预期结果： 提示出现连五一方胜利，并结束游戏
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 输赢结果判断是否正确（是/否）
测试结果： 通过/不通过

4．禁手功能的测试

禁手功能的测试，主要分为两种：三三禁手和四四禁手，如下所示。

（1）三三禁手的测试，主要内容如表 10.7 所示。

表 10.7　三三禁手功能的测试

测试编号：1.7.7	类别：A
项　　目：五子棋游戏测试	
分 项 目：三三禁手的测试	
测试目的：测试五子棋游戏能否对三三禁手进行判断	
测试配置：	
预置条件： 两台 PC 的程序已经通过网络进行连接，并已经开始游戏	
测试步骤： 黑白双方分别落子，当黑方出现三三禁手时	
预期结果： 提示黑方出现禁手，判定白方胜利	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 三三禁手功能的判断是否正确（是/否）	
测试结果： 通过/不通过	

（2）四四禁手的测试，主要内容如表 10.8 所示。

表 10.8　四四禁手功能的测试

测试编号：1.7.8	类别：A
项　　目：五子棋游戏测试	
分 项 目：四四禁手的测试	
测试目的：测试五子棋游戏能否对禁手进行判断	
测试配置：	
预置条件： 两台 PC 的程序已经通过网络进行连接，并已经开始游戏	
测试步骤： 黑白双方分别落子，当黑方出现四四禁手时	
预期结果： 提示黑方出现禁手，判定白方胜利	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 四四禁手功能的判断是否正确（是/否）	
测试结果： 通过/不通过	

10.2　根据用例文档进行测试

编写完成五子棋游戏的测试用例文档后，就可以根据测试用例文档进行功能性测试。在本节中笔者将详细介绍测试过程和结果。

10.2.1　网络连接测试的演示

网络连接的测试项在测试用例文档中编号是 1.7.1～1.7.3。其详细测试操作方法的图解过程如下所示。

1．网络连接主机设置的测试

（1）启动“五子棋游戏”，选中“操作”|“新游戏”菜单项，如图 10.1 所示。

（2）在弹出的对话框中，选中“主机”单选按钮，填写端口号（5001），并单击“确认”按钮，如图 10.2 所示。

（3）再选中“操作”|“新游戏”菜单项。

判断结果：在弹出的对话框中，查看“主机”设置中的“端口号”仍然是“5001”，说明设置是成功的。填写测试用例编号 1.7.1 结果为“通过”。

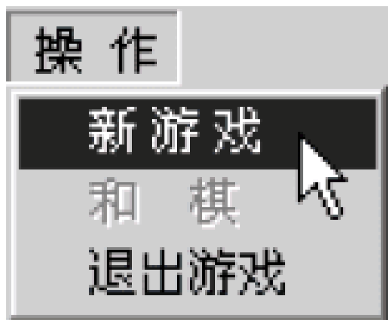


图 10.1　选中“操作”|“新游戏”菜单项

2．网络连接客户机设置的测试

（1）启动“五子棋游戏”，选中“操作”|“新游戏”菜单项，如图 10.1 所示。

（2）在弹出的对话框中，选中“客户机”单选按钮，设置连接主机 IP 地址（192.168.1.67）和端口号（5001），如图 10.3 所示。



图 10.2　主机网络设置对话框



图 10.3　客户机网络设置对话框

（3）再选中“操作”|“新游戏”菜单项。

判断结果：在弹出的对话框中，选中“客户机”选项时，如果连接主机 IP 地址（192.168.1.67）及端口号（5001）和单击“确认”按钮退出前的结果相同，说明设置是成功的。填写测试用例编号 1.7.2 结果为“通过”。

3．网络连接并进行通信的测试

（1）在主机上，设置端口号为“5001”。单击“确认”按钮，等待客户机连接。

（2）在客户机上，设置主机 IP 地址及端口号为“5001”。单击“确定”按钮，连接主机。

（3）当双方连接成功时，弹出“连接成功，可以开始游戏”提示对话框。操作的结果如图 10.4 所示。

（4）黑方可以在棋盘上落子。操作的结果如图 10.5 所示。



图 10.4　开始游戏提示对话框

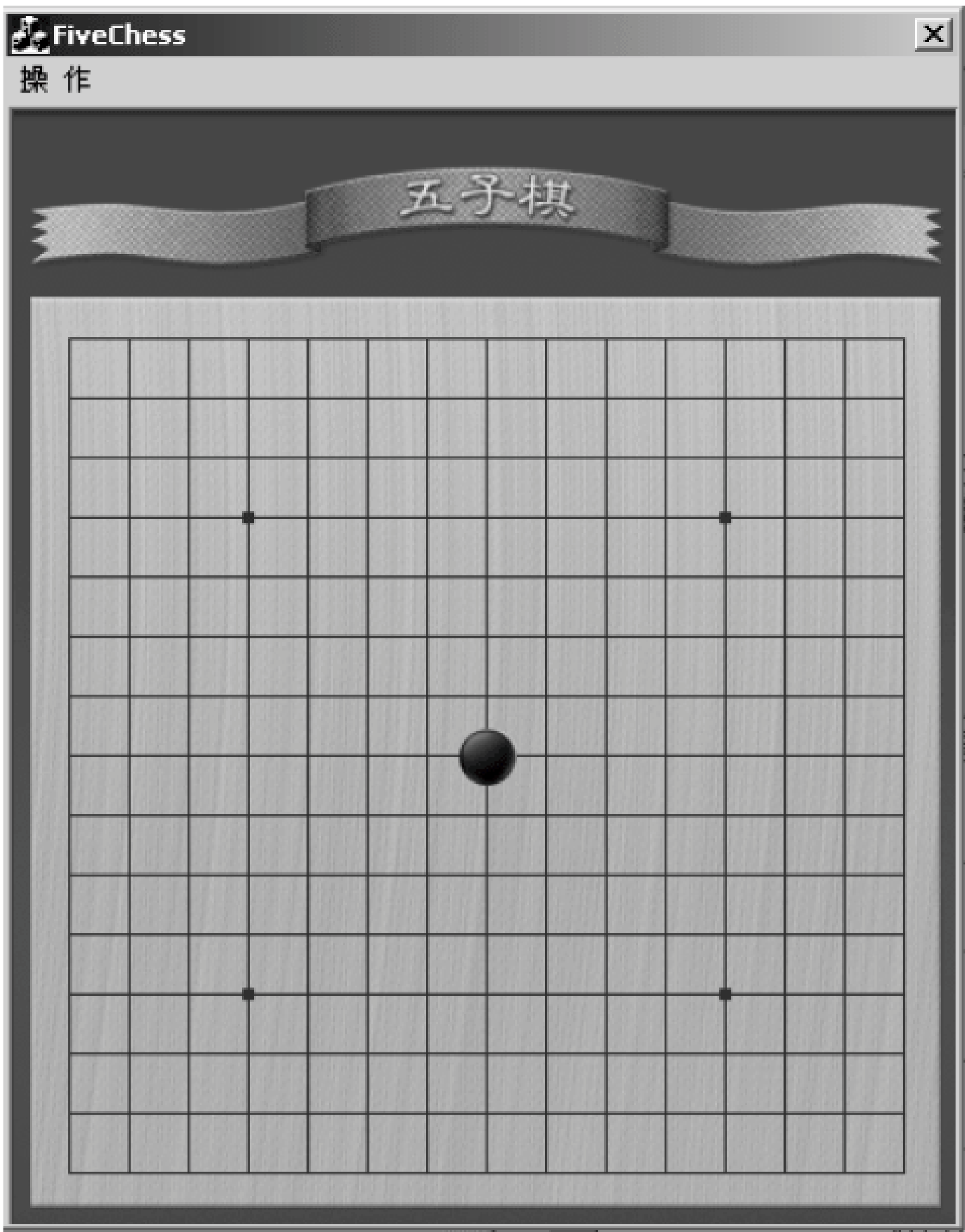


图 10.5　在棋盘上落黑子

判断结果：网络中双机相互连接成功。填写测试用例编号 1.7.3 结果为“通过”。

10.2.2　游戏互动测试的演示

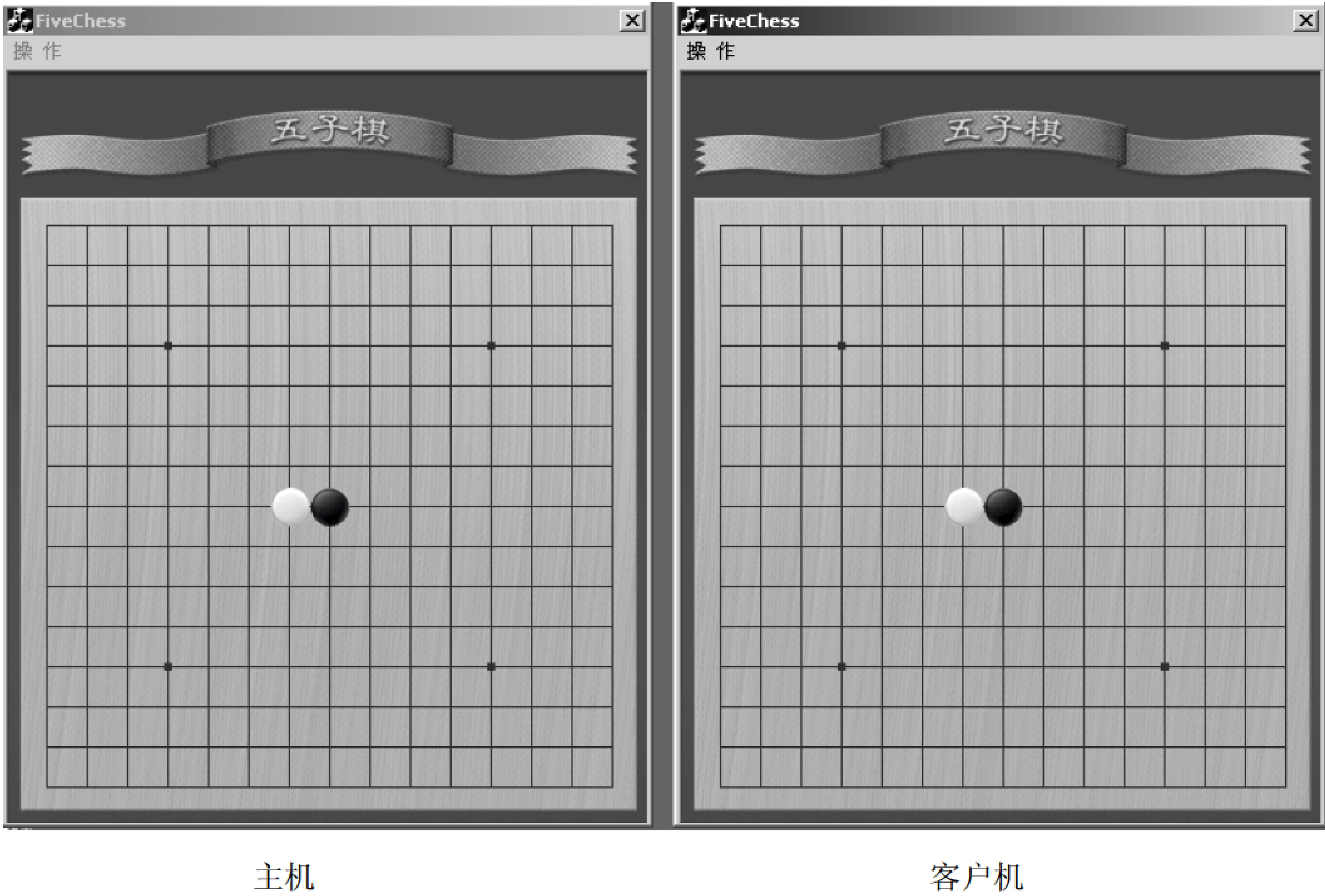
游戏互动测试的测试项在测试用例文档中编号是 1.7.4 和 1.7.5。其详细测试操作方法的图解过程如下所示。

1．落子数据传送的测试

（1）在黑方落子后，查看客户机游戏棋盘上，对应位置是否出现黑子。

（2）在白方落子后，查看主机游戏棋盘上，对应位置是否出现白子。

以上两项测试过程的操作结果如图 10.6 所示。



主机　　　　客户机

图 10.6　落子数据传送操作结果

判断结果：通过实际操作，落子数据的传送成功，填写测试用例编号 1.7.4 结果为“通过”。

2．和棋请求数据传送的测试

（1）由白方发出和棋请求，如图 10.7 所示。

（2）黑方弹出同意或者拒绝请求提示对话框，如图 10.8 所示。

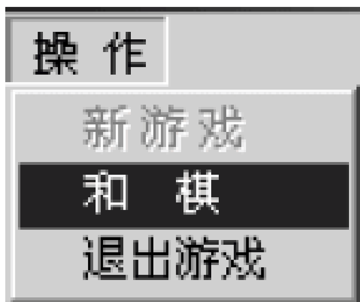


图 10.7　选择“和棋”选项

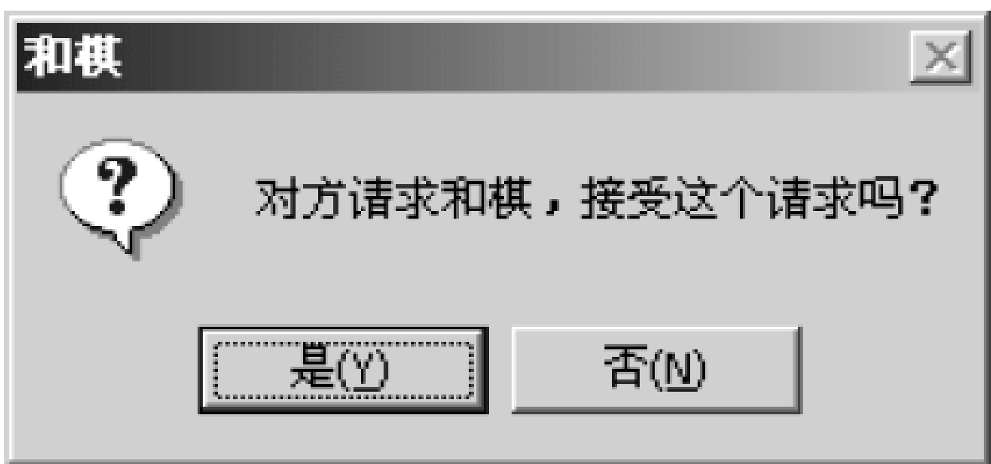


图 10.8　同意或者拒绝请求提示对话框

（3）单击“否（N）”按钮，白方出现如图 10.9 所示对话框。

（4）重复前面第（1）和第（2）步，单击“是（Y）”按钮，白方会出现如图 10.10 所示对话框。



图 10.9　和棋被拒绝提示对话框



图 10.10　和棋请求同意提示对话框

判断结果：通过实际操作，和棋请求数据传送成功，填写测试用例编号 1.7.5 结果为“通过”。

10.2.3　输赢结果测试的演示

输赢结果的测试项在测试用例文档中的编号是 1.7.6。其详细测试操作方法的图解过程如下所示。

（1）开始游戏后，黑白双方分别落子，黑方出现连五时，黑方出现如图 10.11 所示的提示对话框。

（2）白方出现如图 10.12 所示的提示对话框。

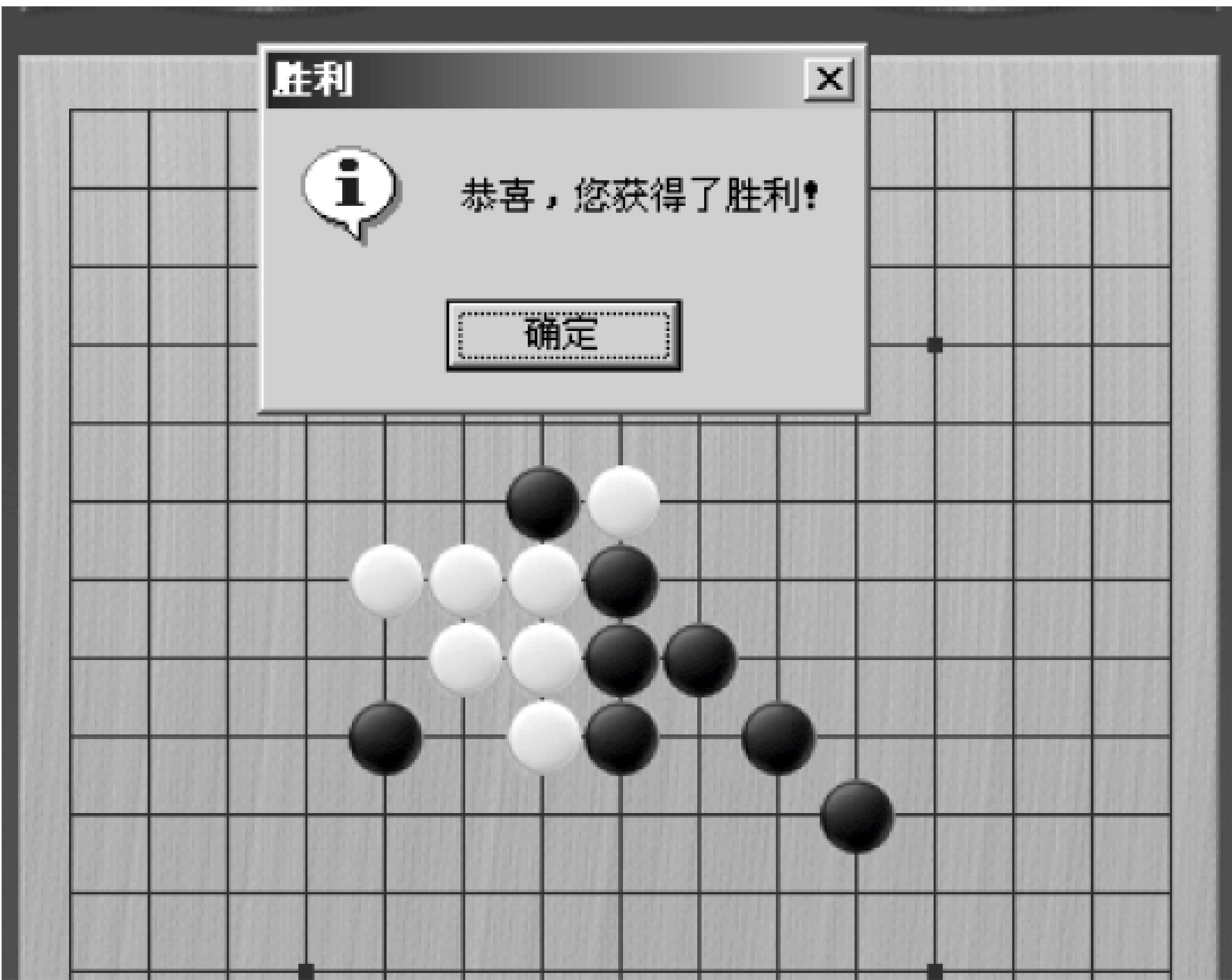


图 10.11　连五出现时黑方的对话框

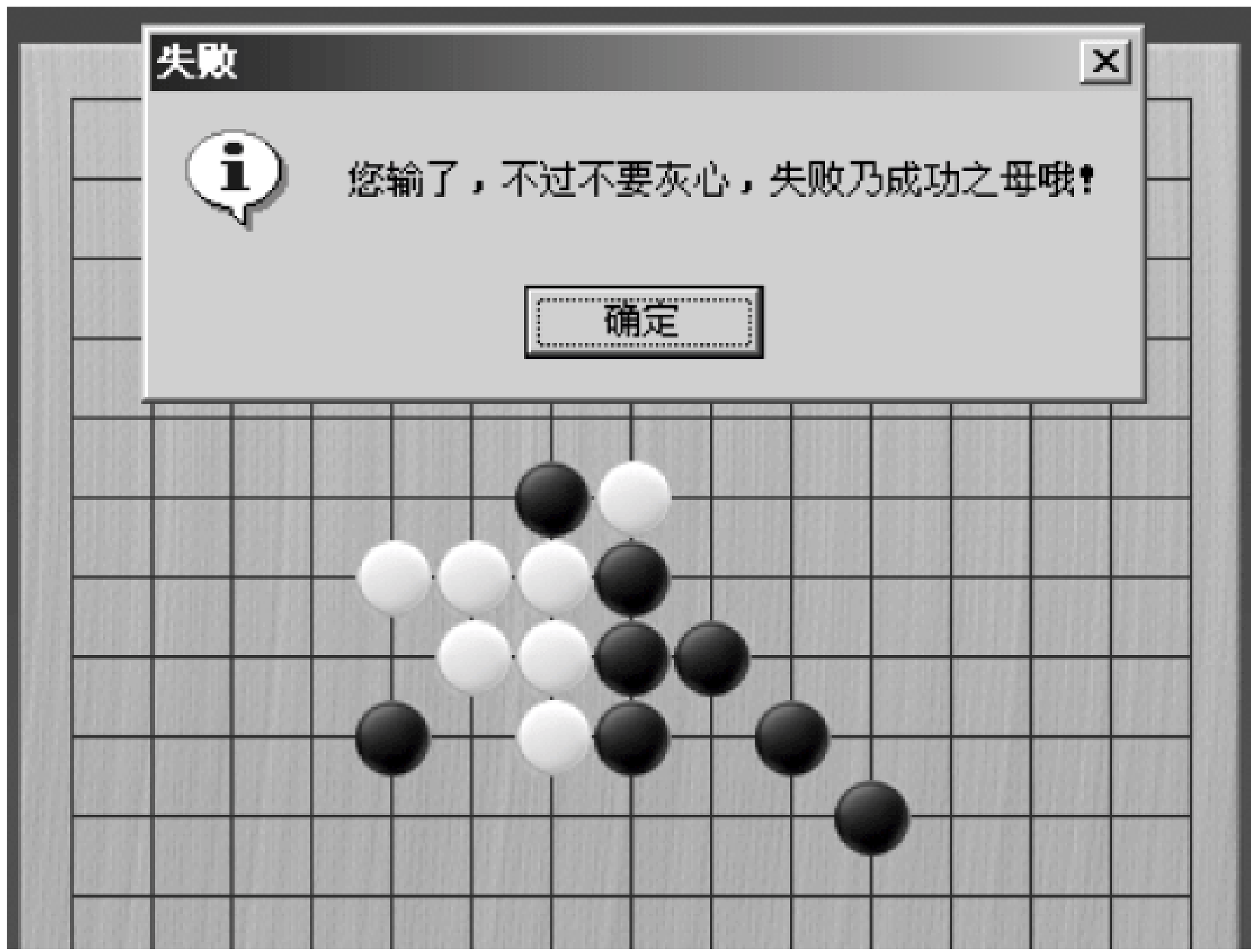


图 10.12　连五出现时白方的对话框

判断结果：通过实际操作，判断输赢结果功能有效，填写测试用例编号 1.7.6 结果为“通过”。

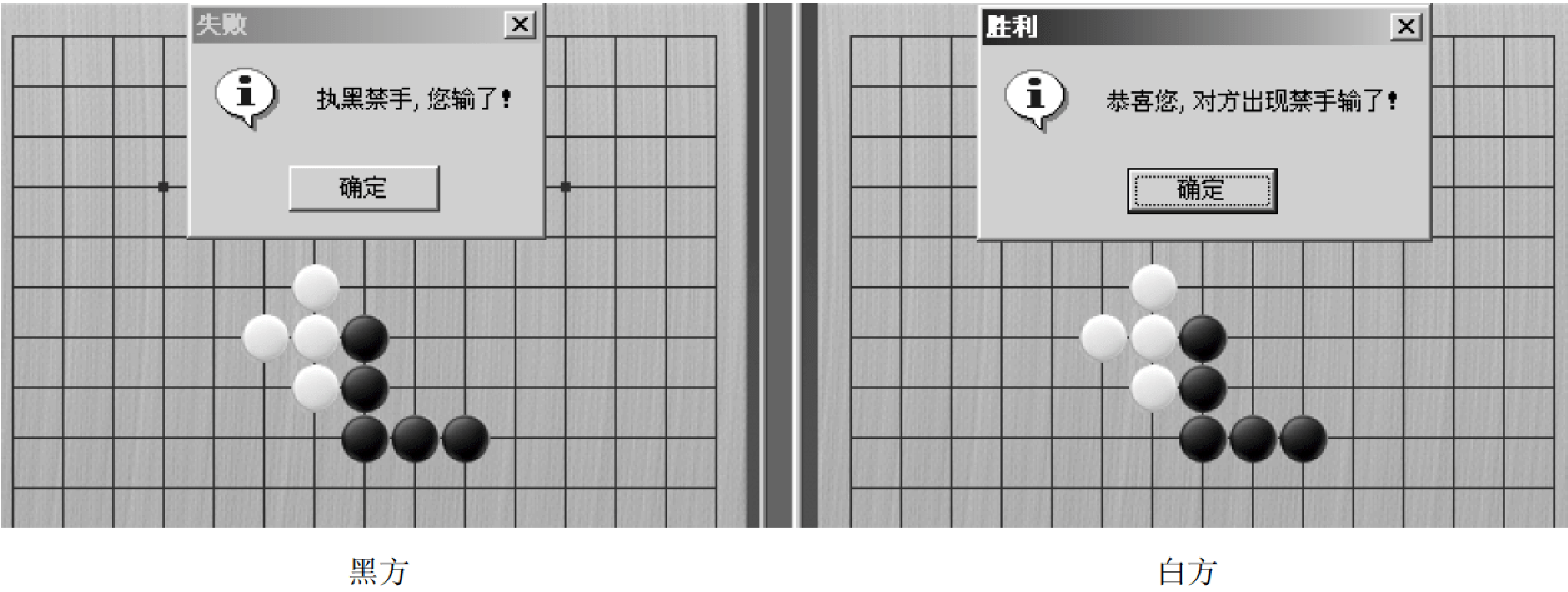
10.2.4　禁手功能测试的演示

禁手功能的测试项在测试用例文档中编号是 1.7.7 和 1.7.8。其详细测试操作方法的图解过程如下所示。

1．三三禁手的测试

（1）开始游戏后，黑白双方分别落子。

（2）当黑方出现三三禁手时，双方出现如图 10.13 所示的提示对话框。



黑方　　　　白方

图 10.13　三三禁手双方出现的提示对话框

判断结果：通过实际操作，三三禁手功能有效，填写测试用例编号 1.7.7 结果为“通过”。

2．四四禁手的测试

（1）开始游戏后，黑白双方分别落子。

（2）当黑方出现四四禁手时，双方出现与图 10.13 同样的提示对话框。

判断结果：通过实际操作，四四禁手功能有效，填写测试用例编号 1.7.8 结果为“通过”。

10.2.5　综合测试结果

通过前面 4 个小节的操作，五子棋游戏整合测试的所有测试项目都通过了测试。说明五子棋游戏基本满足用户需求，已经可以交付某公司使用。

最后提醒读者，上面的测试中，只对一种测试例进行了演示。并不包含全部的测试，例如，禁手测试中，只演示了横竖方向，并没有斜向的测试。而在实际项目中，所有方向都是必须测试的过程。

10.3　总　　结

通过本章的学习，希望读者可以掌握如何编写项目中的测试用例文档，以及如何按照测试用例文档来进行实际测试的方法。

至此，五子棋游戏项目开发全部结束。在第 3 篇中，笔者将讲解更多的实际项目开发案例，来丰富读者的实际开发经验。希望读者能够在学习完成后，充分掌握游戏项目开发的方方面面的实践知识。

第 3 篇　其他游戏开发案例

通过前面内容的学习，各位读者不仅已经了解五子棋游戏项目开发中文档和代码的编写方法，同时也掌握了实际项目开发的流程。唯一缺少的是更多的实际项目开发经验。本篇的主要内容就是为了弥补这种不足，提供多个实际游戏项目案例来丰富读者对游戏项目开发的经验。

本篇是对前一篇内容的扩展，主要分为 6 章，分别是贪吃蛇游戏项目、俄罗斯方块游戏项目、连连看游戏项目、黑白棋游戏项目、扫雷游戏项目和推箱子游戏项目开发。

希望通过本篇的学习，能够丰富读者的游戏项目开发经验，充分掌握游戏设计与实现的完整流程，成为一名优秀的游戏开发工程师。

第 11 章　贪吃蛇游戏项目开发

相信很多的读者都玩过贪吃蛇游戏，本章就是通过某公司贪吃蛇游戏项目的开发实例，来继续介绍如何进行相关游戏项目的开发。

本章主要涉及的内容如下：

- 贪吃蛇项目的需求分析。
- 贪吃蛇游戏的概要设计。
- 贪吃蛇游戏操作界面设计。
- 贪吃蛇游戏的详细设计及代码。
- 贪吃蛇游戏的测试用例文档的编写及测试演示。

11.1　贪吃蛇游戏项目的需求分析

需求获得和分析是项目开发的基础。只有获得明确的需求做出好的需求分析，才是保证项目成功的基础。

技巧：在确定需求时，一定要加强与用户的沟通，最好让用户参与进来。

11.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到贪吃蛇游戏开发的资料如下所述。

1．贪吃蛇游戏的由来

在圣经中，蛇引诱夏娃吃了苹果之后，就被贬为毒虫，是阴险的象征。而且由于蛇吃东西的时候，是将整只动物吞进去的，所以大约在文艺复兴的时候，有人发明了一种游戏，就是现在贪吃蛇的前身，后来慢慢地发展成为今天的贪吃蛇游戏。

2．贪吃蛇游戏的操作方法

游戏通过上、下、左、右方向键控制贪吃蛇移动并吃掉屏幕上出现的果实。

3．贪吃蛇游戏的基本规则

整个贪吃蛇自动向前移动，当吃到果实时就得分，并将身体长度增加一小节。在游戏中不能碰到墙壁和自己的身体。当碰到墙壁和自己的身体时，宣告贪吃蛇死亡，并结束当前游戏。然后记录当前分数。

4．英雄榜的显示及更新

当有玩家得到的分数超过当前记录分数线时，就把分数保存下来，在结束游戏时，要求玩家把名字保存下来。游戏初始时记录分数线为 0。例如，当第一个玩家得分为 10 分，那么结束游戏时，这个玩家的记录分将被保存下来并作为记录分数线。直到有玩家的得分超过 10 分，才能更新当前记录分数线并在退出游戏时保存玩家分数及名字。

5．游戏难度可以选择

在游戏开始前，可以选择贪吃蛇移动的速度，速度越快吃到果实的得分也就越高。相应的游戏难度也就越高。难度分为低、中、高 3 种。

6．游戏可以选择播放背景音乐

在游戏开始后，可以选择播放背景音乐。

7．游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

11.1.2　对语言描述进行需求分析

在 11.1.1 节中，对用户的需求进行了描述，但这些描述并不是很直观。所以必须对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

说明：需求分析文档要从程序员的角度编写，突出用户语言描述需求中的要点。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类贪吃蛇游戏。特制定本说明书来用于描述某公司贪吃蛇项目开发的功能性需求。

1.1　编写目的

使用技术性语言对某公司的贪吃蛇游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含某公司贪吃蛇游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司贪吃蛇游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

《贪吃蛇游戏用户描述文档》。

5．游戏具有的功能

5.1　能够实现贪吃蛇自动向前移动

根据时间间隔，每一次将贪吃蛇的每节身体分别向前移动一格。移动方向为贪吃蛇当前行走方向。

5.2　对游戏中的规则进行判断

当在游戏中贪吃蛇碰到墙壁和自己的身体时，宣告贪吃蛇死亡，并结束当前游戏。然后记录当前分数。

5.3　贪吃蛇的操作

游戏通过键盘上的上、下、左、右 4 个方向键控制贪吃蛇当前行走方向，吃掉屏幕上出现的果实。每吃掉一个果实，蛇身长度增加一节。

5.4　果实的出现

在游戏中，果实的出现，应采用随机方式。当前一个果实被吃掉时，屏幕上随机出现另一个果实。但要注意，果实不应该出现在贪吃蛇的身体所占用的范围内。

5.5　游戏分数的统计方法

当贪吃蛇吃到果实时就得把当前玩家分数增加并将身体长度增加一小节。游戏中的分数的统计方法，采用如下公式进行计算：

```
贪吃蛇身体长度 × 等级 = 分数
```

其中，贪吃蛇身体长度是由吃掉的果实多少决定的。等级分为 3 级，其分值如表 11.1 所示。

表 11.1　贪吃蛇等级划分

等　　级	分　　数
低	2
中	4
高	6

5.6　英雄榜的更新

当有玩家得到的分数超过当前记录分数线时，就把分数保存下来，在结束游戏时，要求玩家把名字保存下来。游戏初始时记录分数线为 0。

例如，第一个玩家得分为 10 分，当他结束游戏时，他的记录分将被保存下来并作为记录分数线。直到有玩家的得分超过 10 分，才能更新当前记录分数线，并在退出游戏时保存玩家的分数及名字。

5.7　游戏难度可以选择

通过主菜单，让玩家在游戏开始前，可以选择贪吃蛇游戏的难度。难度越高，贪吃蛇移动的速度越快，吃到果实的得分也就越高。

5.8　游戏支持背景音乐功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.9　游戏提供帮助说明

在游戏菜单中，提供一个使用说明项，以方便对本游戏不了解的玩家对游戏进行操作和使用。

技巧：只有用户确定了的需求分析文档，才是真实有效的文档。

11.2　贪吃蛇游戏概要设计

概要设计对于程序的开发至关重要。下面笔者将开始介绍贪吃蛇游戏项目概要设计是如何编写的。

概要设计是对整个游戏的功能进行概念性的设计，不涉及详细的内容，只给出大体的功能性框架即可。笔者编写的贪吃蛇游戏的概要设计文档内容如下所述。

技巧：对于整体框架的设计一定要大而全，忽略其中的细节部分。

1．引言

1.1　编写目的

为了让各个开发人员明白贪吃蛇游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．术语

3．参考文献

《贪吃蛇游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高键盘操作易用性。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。
- Microsoft® Visual C++ 6.0。

4.3　需求概述

参见《贪吃蛇游戏需求分析说明书》。

4.4　条件与限制

无。

5．总体设计

5.1　贪吃蛇游戏的功能架构（如图 11.1 所示）

5.2　各功能处理流程

内容参见《贪吃蛇游戏各功能详细设计文档》。

6．接口设计

内容参见《贪吃蛇游戏操作界面设计文档》。

7．类结构设计

游戏由 6 个类组成，如图 11.2 所示。

- 游戏规则类：主要负责各种类的调用及游戏规则的实现。
- 主游戏类：主要负责贪吃蛇及果实的更新和显示。
- 蛇身操作类：主要负责蛇身的移动、增加及移动方向。

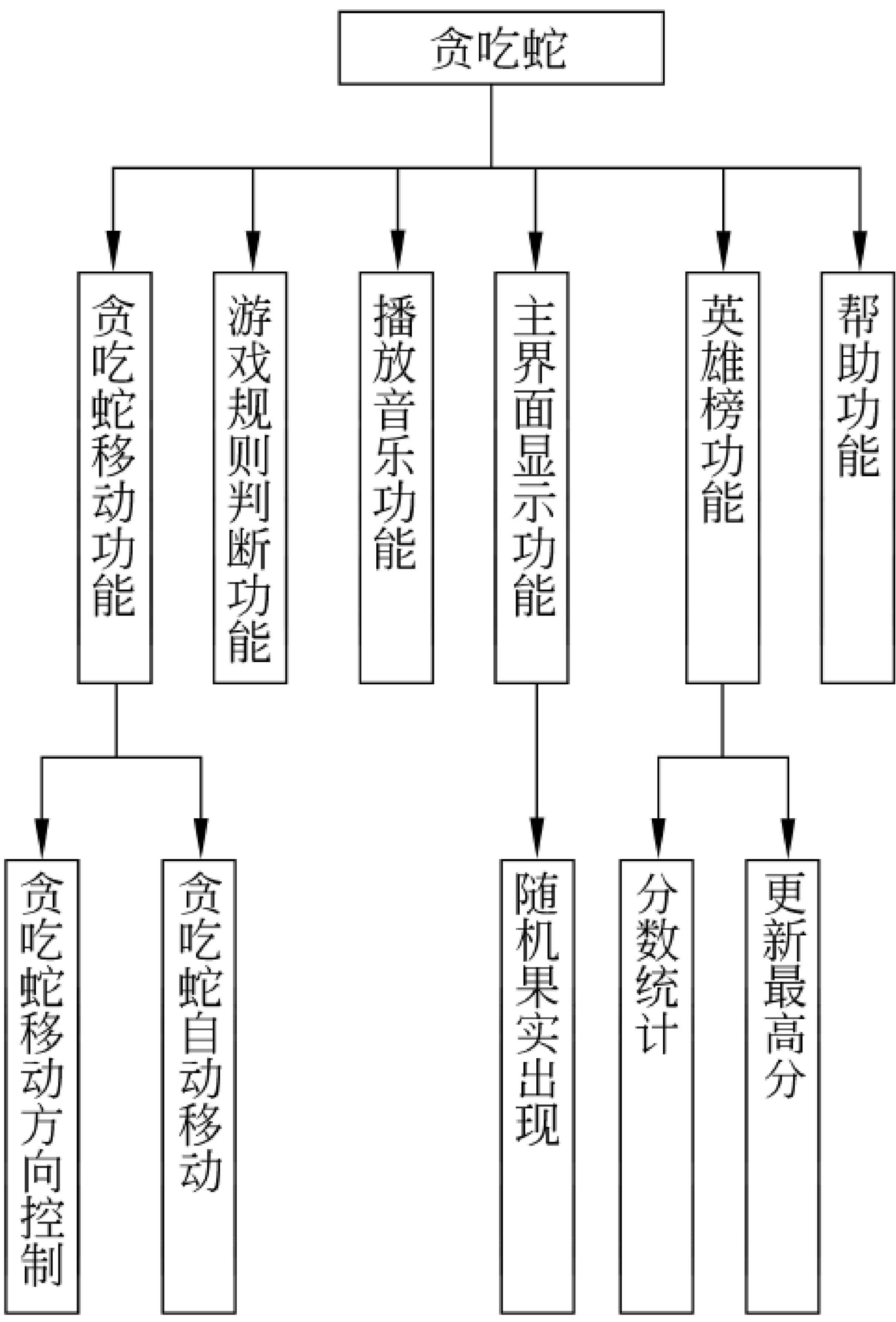


图 11.1　贪吃蛇游戏功能架构

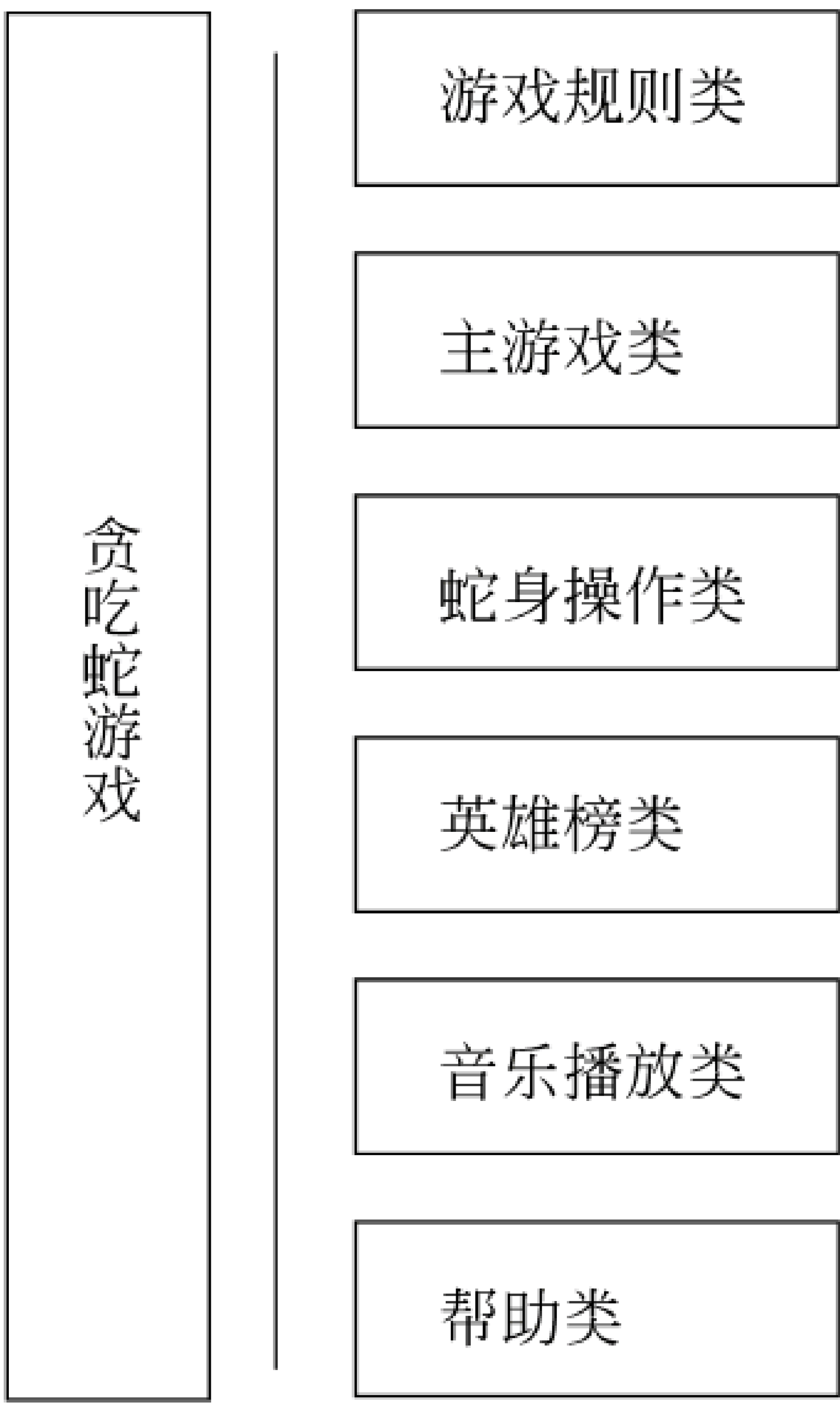


图 11.2　游戏主要类结构

- 英雄榜类：主要负责游戏分数的统计及高分记录的更新。
- 音乐播放类：主要负责游戏中背景音乐的播放。
- 帮助类：主要负责帮助提示的显示及其他辅助信息。

8．出错处理设计

8.1　出错输出信息

当游戏中出现错误时，采用弹出对话框的方式来提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误时，采用中止当前游戏并重新开始新游戏的方法来处理游戏中的错误。

9．维护设计

由于整个贪吃蛇游戏项目在开发完成后，基本不会有太多的变动，所以维护的主要任务是把用户使用中遇到的错误进行解决就行。

11.3　贪吃蛇游戏操作界面及测试用例设计

在编写完概要设计之后，就可以根据概要设计来开始编写《游戏操作界面设计文档》和《测试用例文档》。

11.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的贪吃蛇游戏的操作界面设计文档内容如下所述。

技巧：在界面文档完成后，一定要提交用户查阅，让用户知道游戏界面的排列。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确游戏的操作界面是如何设计的，特制定本文档用于描述本公司贪吃蛇游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含本公司贪吃蛇游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《贪吃蛇游戏需求分析说明书》和《贪吃蛇游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面设计

游戏的主界面设计如图 11.3 所示。

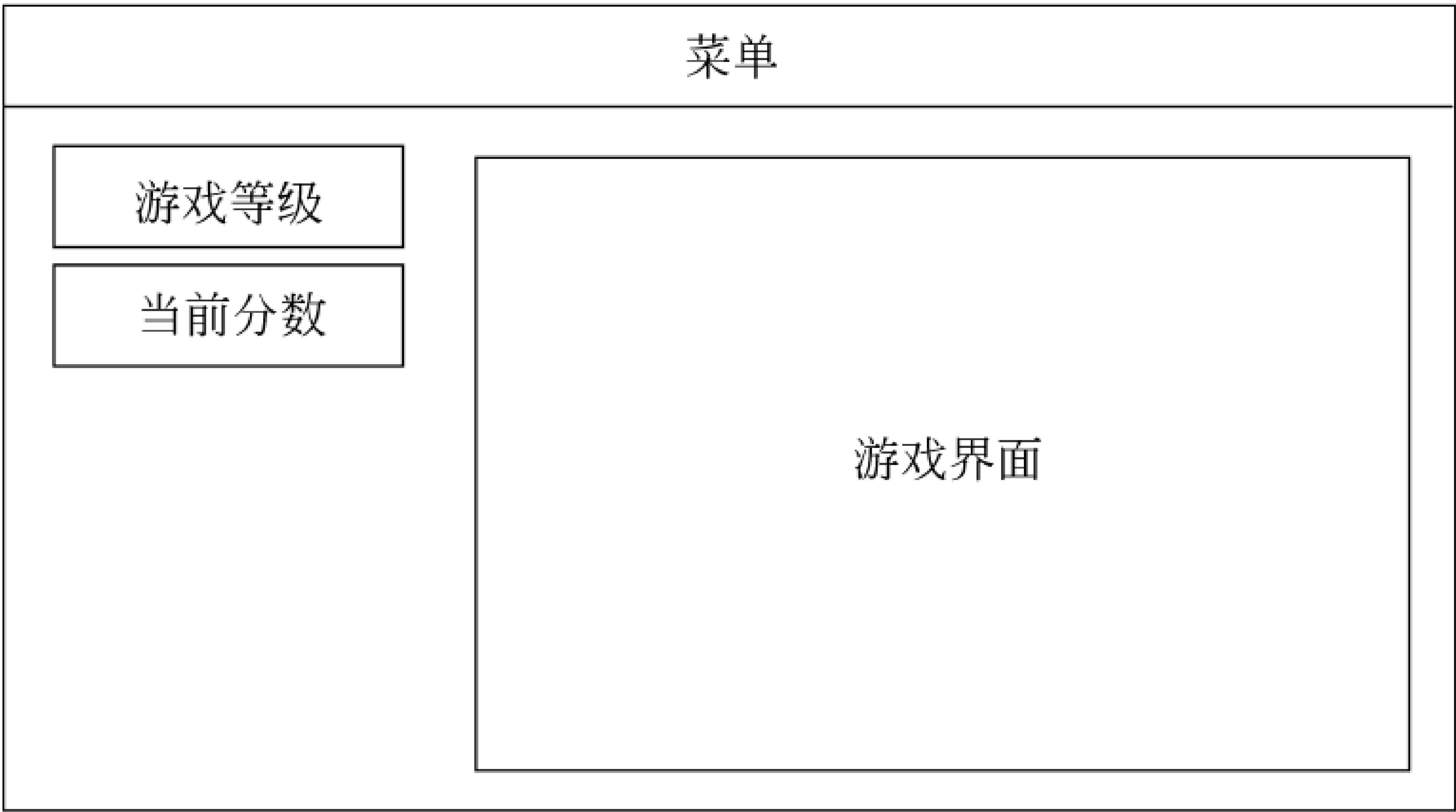


图 11.3　贪吃蛇游戏的主界面结构

5.2　游戏的菜单设计

贪吃蛇游戏的菜单结构设计如图 11.4 所示。

5.3　英雄榜对话框设计

贪吃蛇游戏中的英雄榜对话框设计，如图 11.5 所示。

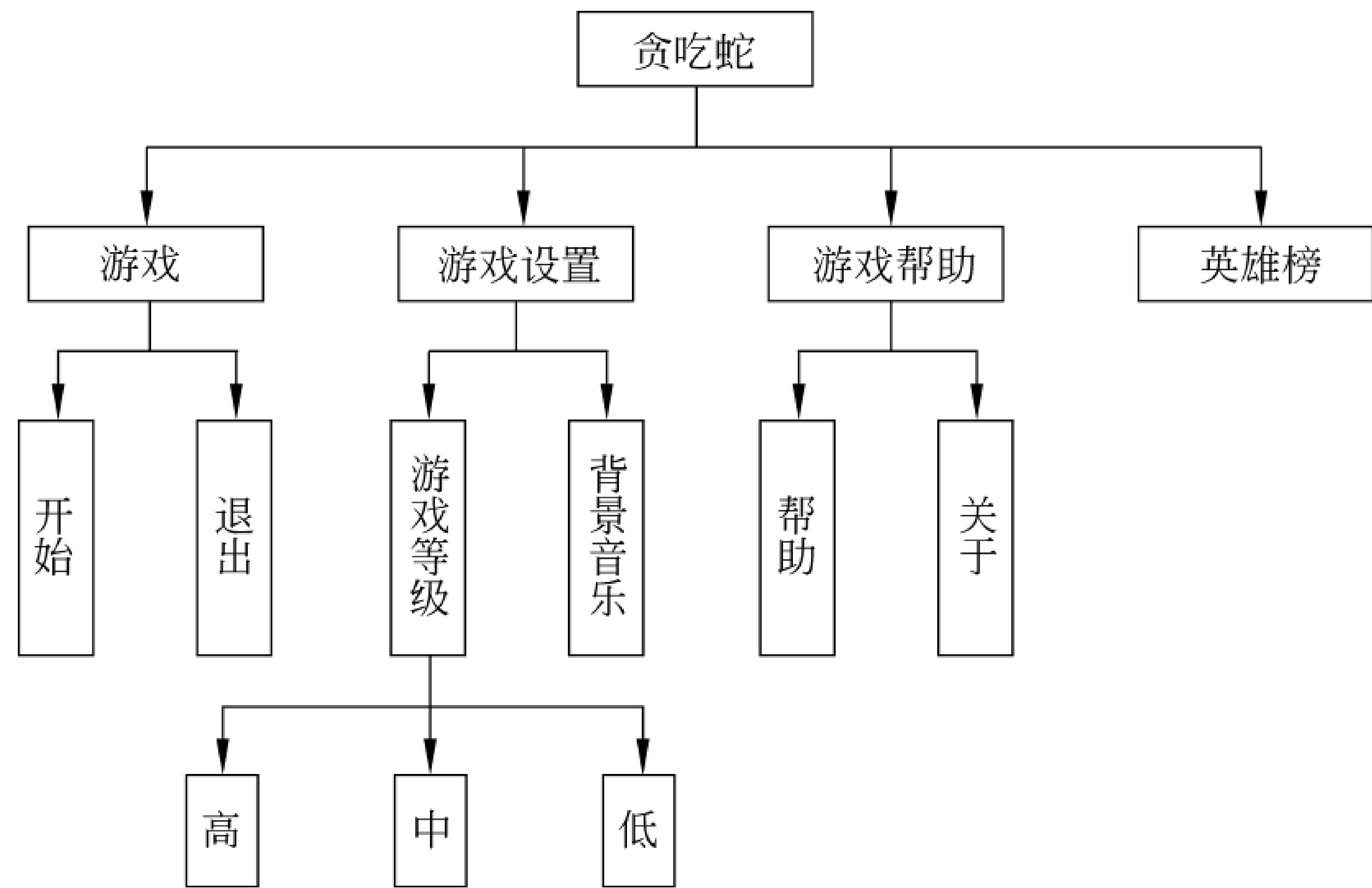


图 11.4　贪吃蛇游戏的菜单结构

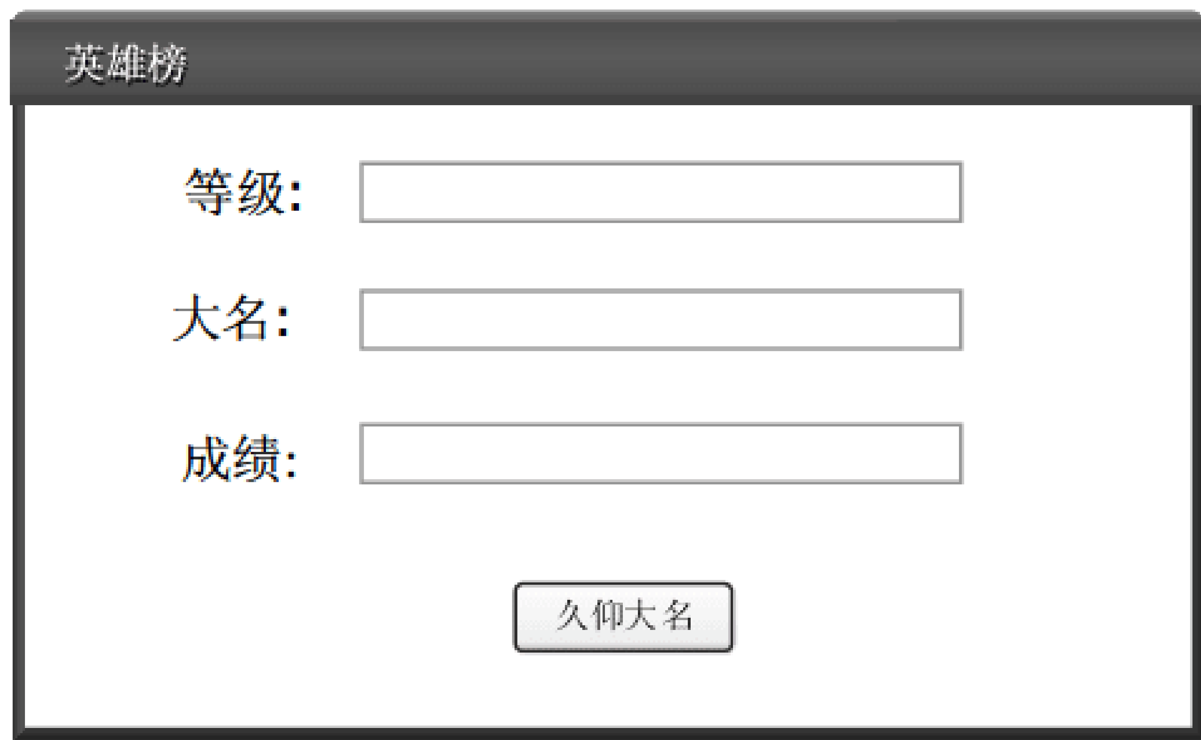


图 11.5　英雄榜界面

11.3.2　测试用例文档

测试用例文档在概要设计之后，就可以开始编写了。其主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的贪吃蛇游戏测试用例文档内容如下所述。

技巧：测试用例文档要力争做到覆盖面越广越好。

1．引言

本文档主要用于某公司在开展贪吃蛇游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了贪吃蛇游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过，方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的贪吃蛇游戏项目的测试内容和测试方法提出规定。原则上

只能在本公司内部使用，用于指导本公司的测试人员，进行贪吃蛇游戏项目的测试和验收。

3．使用对象

本测试用例文档使用对象主要是与某公司贪吃蛇游戏开发相关的需求分析、测试和维护等部门（单位）的人员。

4．参考文献

《贪吃蛇游戏的需求分析说明书》；

《贪吃蛇游戏的概要设计文档》；

《贪吃蛇游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试项目

测试项目主要针对贪吃蛇中的各种功能进行整合性测试，共包含如下几个项目。

（1）游戏等级设置的测试，其主要内容如表 11.2 所示。

表 11.2　游戏等级设置的测试

测试编号：1.7.1	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：游戏等级设置的测试	
测试目的：测试贪吃蛇游戏能否进行游戏等级的设置	
测试配置：	
预置条件： 贪吃蛇游戏源程序已经编译完成，并可以运行	
测试步骤： 选中“游戏设置”|“游戏等级”菜单项。 在弹出的下级菜单中，分别依次选中“低”、“中”、“高”等级	
预期结果： 在游戏主界面上，显示的游戏等级是否与选择的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏等级是否可以设置（是/否）	
测试结果： 通过/不通过	

（2）贪吃蛇移动功能的测试，其主要内容如表 11.3 所示。

表 11.3　贪吃蛇移动功能的测试

测试编号：1.7.2	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：贪吃蛇移动功能的测试	
测试目的：测试游戏中贪吃蛇是否能够按指定方向移动	
测试配置：	

续表

预置条件： 贪吃蛇游戏已经开始
测试步骤： 依次单击键盘按键“右”、“上”、“左”、“下”
预期结果： 当单击“右”按键时，贪吃蛇的身体向右移动； 当单击“上”按键时，贪吃蛇的身体向上移动； 当单击“左”按键时，贪吃蛇的身体向左移动； 当单击“下”按键时，贪吃蛇的身体向下移动
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏中贪吃蛇是否能够按指定方向移动（是/否）
测试结果： 通过/不通过

（3）游戏分数统计功能的测试，其主要内容如表 11.4 所示。

表 11.4　游戏分数统计功能的测试

测试编号：1.7.3	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：游戏分数统计功能的测试	
测试目的：测试游戏分数统计功能是否正确	
测试配置：	
预置条件： 贪吃蛇游戏源程序已经编译完成，并可以运行	
测试步骤： 分别选择游戏等级为“低”、“中”、“高”； 开始游戏，并且已经吃到两个果实	
预期结果： “低”等级时，游戏分数显示为“4”； “中”等级时，游戏分数显示为“8”； “高”等级时，游戏分数显示为“12”	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 测试游戏分数统计功能是否正确（是/否）	
测试结果： 通过/不通过	

（4）果实随机出现功能的测试，其主要内容如表 11.5 所示。

表 11.5　果实随机出现功能的测试

测试编号：1.7.4	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：果实随机出现功能的测试	
测试目的：测试游戏中的果实出现的位置是否是随机的，并且不在蛇身范围内	
测试配置：	
预置条件： 键盘准备好； 游戏已经开始； 游戏等级已经设定，并且出现果实	
测试步骤： 吃掉当前屏幕上的果实，查看果实出现的位置； 再吃掉第二次出现的果实，查看果实再次出现的位置； 再吃掉第三次出现的果实，查看果实第三次出现的位置	
预期结果： 果实是随机出现的，与前一次的位置不同； 出现的位置不在蛇身范围内	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的果实出现的位置是否是随机的（是/否）	
测试结果： 通过/不通过	

（5）游戏规则功能的测试，其主要内容如表 11.6 所示。

表 11.6　游戏规则功能的测试

测试编号：1.7.5	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：游戏规则功能的测试	
测试目的：测试贪吃蛇游戏能否对蛇死亡的规则进行判断	
测试配置：	
预置条件： 游戏已经开始； 游戏等级已经设定； 操作贪吃蛇已经吃到若干个果实	
测试步骤： 让贪吃蛇移向边缘，并碰撞墙壁； 让贪吃蛇碰撞自己的身体	
预期结果： 提示游戏结束	

续表

判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 贪吃蛇游戏能否正确对蛇死亡的规则进行判断（是/否）
测试结果： 通过/不通过

（6）背景音乐播放功能的测试，其主要内容如表 11.7 所示。

表 11.7　背景音乐播放功能的测试

测试编号：1.7.6	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试贪吃蛇游戏能否支持播放背景音乐	
测试配置：	
预置条件： 贪吃蛇游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 贪吃蛇游戏能否支持播放背景音乐（是/否）	
测试结果： 通过/不通过	

（7）帮助功能的测试，其主要内容如表 11.8 所示。

表 11.8　帮助功能的测试

测试编号：1.7.7	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：帮助功能的测试	
测试目的：测试贪吃蛇游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选中“游戏帮助”|“帮助”菜单栏	

续表

预期结果： 出现游戏帮助提示，说明游戏操作方法
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 贪吃蛇游戏是否有帮助提示功能（是/否）
测试结果： 通过/不通过

（8）英雄榜功能的测试，其主要内容如表 11.9 所示。

表 11.9　英雄榜功能的测试

测试编号：1.7.8	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：英雄榜功能的测试	
测试目的：测试贪吃蛇游戏的英雄榜记录是否正确	
测试配置：	
预置条件： 玩家在游戏中已经超过上次最高记录分数	
测试步骤： 等待游戏结束； 在弹出的对话框中输入玩家的大名； 选中“英雄榜”菜单	
预期结果： 在弹出的对话框中，查看等级、大名及分数是否被记录	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 贪吃蛇游戏的英雄榜记录是否正确（是/否）	
测试结果： 通过/不通过	

（9）主界面显示功能的测试，其主要内容如表 11.10 所示。

表 11.10　主界面显示功能的测试

测试编号：1.7.9	类别：A
项　　目：贪吃蛇游戏测试	
分 项 目：主界面显示功能的测试	
测试目的：测试主界面是否正确显示背景和贪吃蛇	
测试配置：	
预置条件： 贪吃蛇游戏已经运行	

续表

测试步骤： 游戏运行后，查看主界面背景及贪吃蛇本身是否成功显示
预期结果： 游戏背景及贪吃蛇成功显示
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 主界面是否正确显示背景和贪吃蛇（是/否）
测试结果： 通过/不通过

11.4　贪吃蛇游戏的详细设计

在详细设计文档中，包含了各个功能模块的功能说明及详细流程图，以方便游戏程序开发人员根据文档进行游戏的设计。在这里还是按照前面讲解五子棋游戏时的使用方法，只给出文档中的重点内容。不在这里详细描述设计文档的格式。

技巧：详细设计中要把设计的步骤进行详细描述。

11.4.1　游戏各功能的设计描述

在贪吃蛇游戏中，大致可以分为 5 个功能模块。如下所示。

1．游戏规则模块的算法设计

游戏规则模块的算法主要分为如下几步：

（1）当蛇身每移动一步时，就对贪吃蛇的头部坐标进行判断。

（2）如果已经与界面的边界坐标或者与贪吃蛇 BODY 向量中的坐标重合，说明贪吃蛇已经碰到墙壁或者身体，这时就弹出游戏结束提示。

（3）设置游戏状态为结束状态。

2．蛇身操作模块的算法设计

蛇身操作模块的算法主要分为如下几步：

（1）得到当前按下的移动方向。

（2）如果遇到果实，则 BODY 向量增加一个元素。如果没有遇到果实，不更新 BODY 向量中的元素。

（3）更新贪吃蛇 BODY 向量中的坐标数据。

3．英雄榜模块的算法设计

英雄榜模块的算法主要分为如下几步：

（1）读取配置文件，得到并显示当前最高分记录、大名及等级。

（2）在用户结束游戏时，比较当前用户得分和最高分。如果高于最高分，就弹出“英雄榜”对话框，要求用户输入大名，并连同用户的等级和分数保存到配置文件中。

4．音乐播放模块的算法设计

音乐播放模块比较简单，只需要在用户选择音乐播放时，把音乐资源载入程序并播放。

5．帮助类模块的算法设计

帮助类模块的算法也比较简单，只是把相应的对话框资源显示出来即可。

11.4.2　游戏各功能的流程图

贪吃蛇游戏的各功能流程如图 11.6 所示。

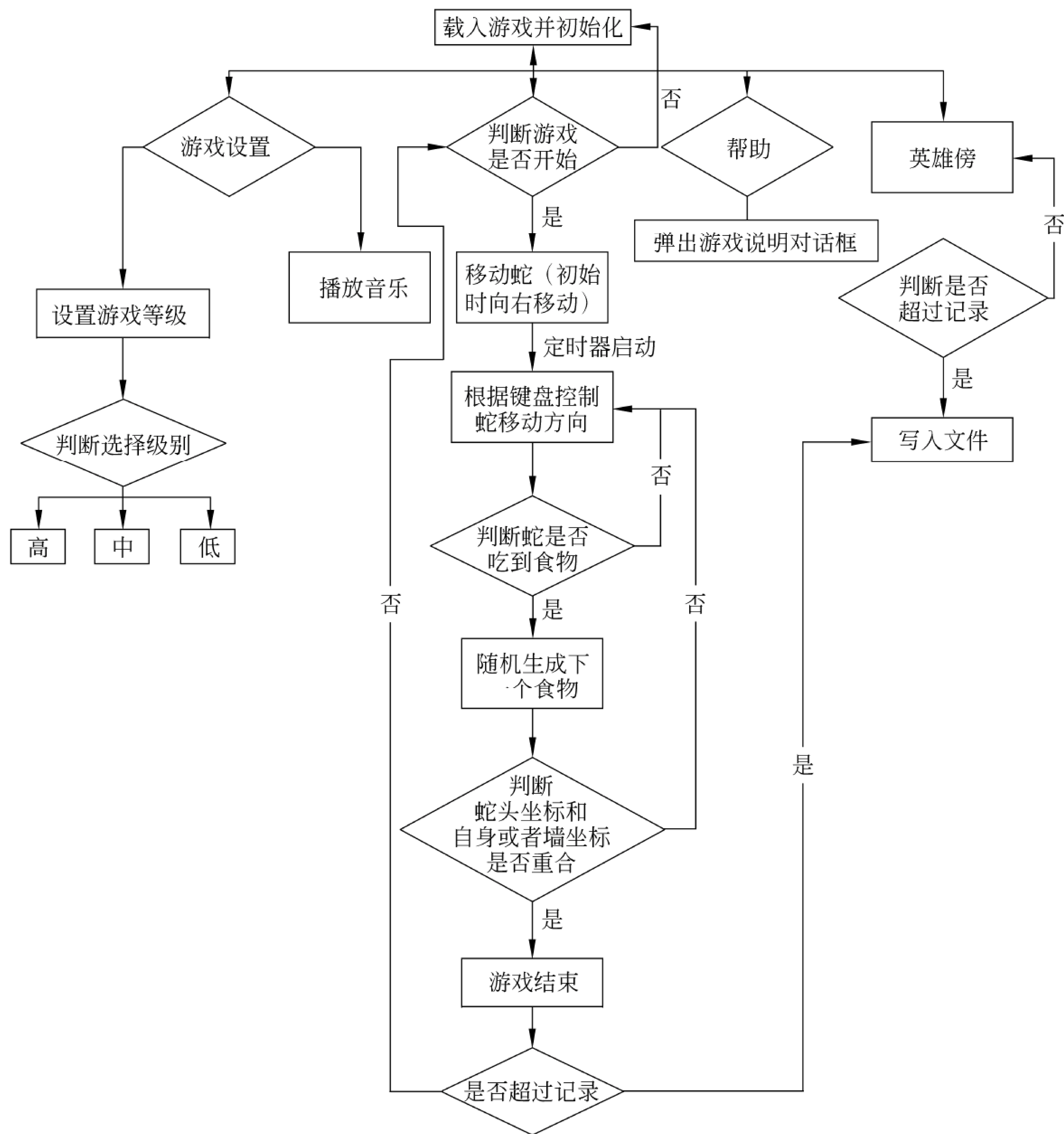


图 11.6　贪吃蛇游戏功能流程

11.5　贪吃蛇游戏界面的实现

贪吃蛇游戏的 Visual C++工程为了使用方便，仍然采用对话框模式来实现。本节主要讲解如何用代码来实现贪吃蛇游戏的各个功能模块。

11.5.1　游戏菜单的实现

在贪吃蛇游戏中，游戏菜单采用的是 Windows 标准菜单实现。所以只需要通过如下几步即可实现游戏的菜单。

（1）在贪吃蛇游戏工程的资源中添加一个菜单资源，其属性如表 11.11 所示。

表 11.11　菜单资源属性

ID	类　　别	说　　明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
IDR_EXIT_GAME	菜单栏	退出游戏
IDR_GAME_LEVEL	弹出菜单	游戏等级
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于
IDR_HERO_LIST	菜单栏	英雄榜

（2）给每个菜单栏添加响应函数，如图 11.7 所示。

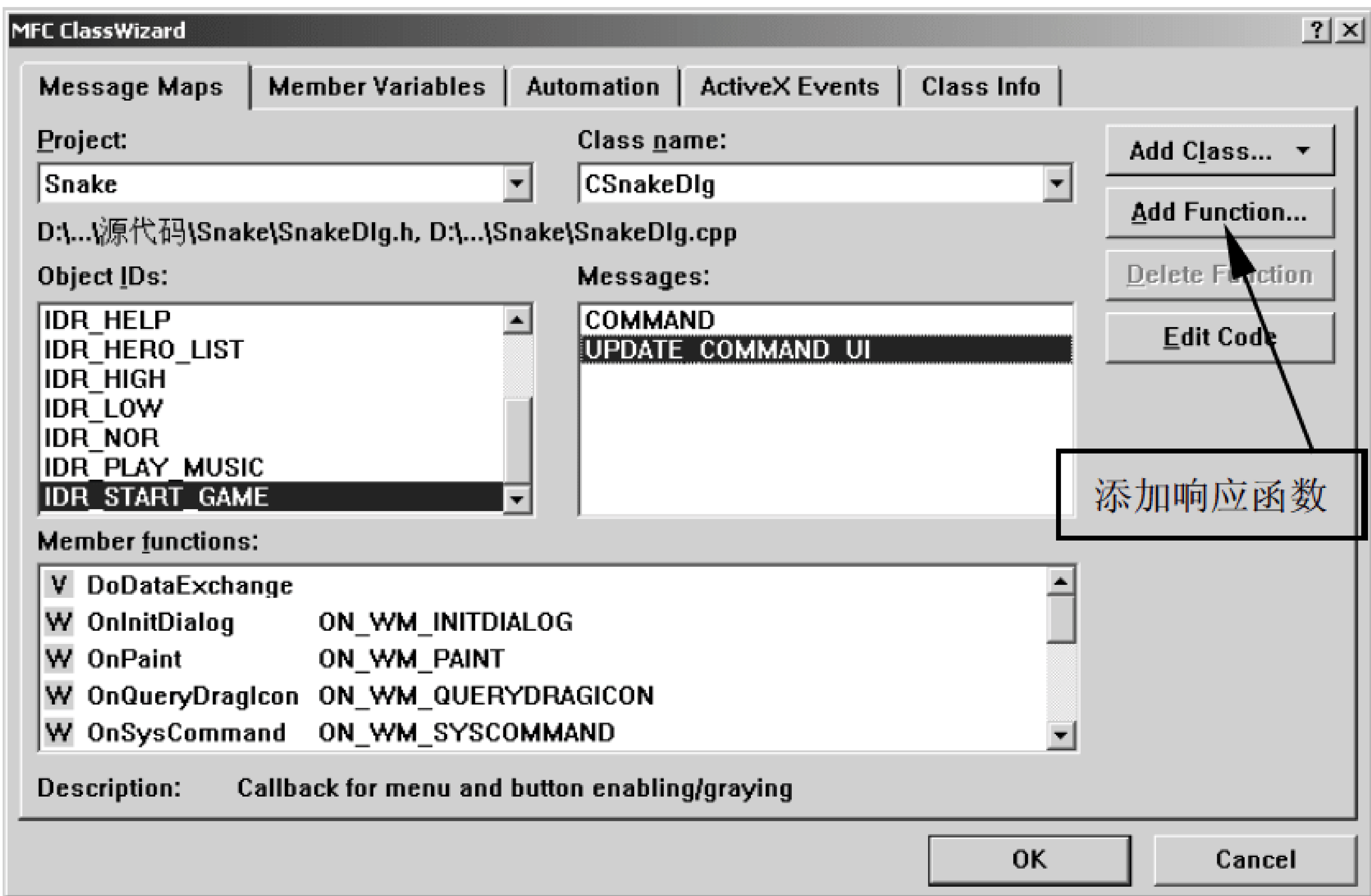


图 11.7　添加菜单响应函数

（3）菜单响应函数的声明，如代码 11.1 所示。

代码 11.1　菜单响应函数的声明

```
01  // SnakeDlg.h 对话框头文件
02
03
04  #if !defined(AFX_SNAKEDLG_H__)
05  #define AFX_SNAKEDLG_H__
06
07  ////////////////////////////////////////////////////////////////////
08  // CSnakeDlg dialog 的声明
09
10  class CSnakeDlg : public CDialog
11  {
12  public:
13      void Help();                            //帮助功能函数
14      void HeroList();                        //英雄榜功能函数
15      void StartGame();                       //开始游戏功能函数
16      void PlayBackMusic(BOOL bflg);          //播放背景音乐功能函数
17      void InitMenu();                        //初始化菜单
18      CSnakeDlg(CWnd* pParent = NULL);        //默认构造函数
19
20      enum { IDD = IDD_SNAKE_DIALOG };
21
22      protected:
23      virtual void DoDataExchange(CDataExchange* pDX);
                                                // DDX/DDV support
24
25  protected:
26      HICON m_hIcon;                          //主图标
27      CMenu m_main_menu;                      //主菜单对象
28      virtual BOOL OnInitDialog();
29      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
30      afx_msg void OnPaint();
31      afx_msg HCURSOR OnQueryDragIcon();
32      afx_msg void OnUpdatePlayMusic(CCmdUI* pCmdUI);
33      afx_msg void OnUpdateHigh(CCmdUI* pCmdUI);
34      afx_msg void OnUpdateLow(CCmdUI* pCmdUI);
35      afx_msg void OnUpdateNor(CCmdUI* pCmdUI);
36      afx_msg void OnUpdateHelp(CCmdUI* pCmdUI);
37      afx_msg void OnUpdateHeroList(CCmdUI* pCmdUI);
38      afx_msg void OnUpdateExitGame(CCmdUI* pCmdUI);
39      afx_msg void OnUpdateAbout(CCmdUI* pCmdUI);
40      afx_msg void OnUpdateStartGame(CCmdUI* pCmdUI);
41      DECLARE_MESSAGE_MAP()
42  };
43
44  #endif // !defined(AFX_SNAKEDLG_H__)
```

（4）菜单响应函数是通过玩家对菜单栏的选择，实现与游戏程序进行交互接口。菜单响应函数中主要通过调用其他成员函数来实现功能，减少直接处理的过程，如代码 11.2 所示。

代码 11.2　菜单响应函数的实现

```
01  // SnakeDlg.cpp 对话框源文件
02  #include "stdafx.h"
03  #include "Snake.h"
```

```
04  #include "SnakeDlg.h"
05  ...                                     //省略部分代码，请查阅光盘源文件
06  //函数映射表
07  BEGIN_MESSAGE_MAP(CSnakeDlg, CDialog)
08      ON_WM_SYSCOMMAND()
09      ON_WM_PAINT()
10      ON_WM_QUERYDRAGICON()
11                                          //播放背景音乐菜单栏响应函数
12      ON_UPDATE_COMMAND_UI(IDR_PLAY_MUSIC, OnUpdatePlayMusic)
13                                          //游戏等级-高菜单栏响应函数
14      ON_UPDATE_COMMAND_UI(IDR_HIGH, OnUpdateHigh)
15                                          //游戏等级-低菜单栏响应函数
16      ON_UPDATE_COMMAND_UI(IDR_LOW, OnUpdateLow)
17                                          //游戏等级-中菜单栏响应函数
18      ON_UPDATE_COMMAND_UI(IDR_NOR, OnUpdateNor)
19                                          //帮助菜单栏响应函数
20      ON_UPDATE_COMMAND_UI(IDR_HELP, OnUpdateHelp)
21                                          //英雄榜菜单栏响应函数
22      ON_UPDATE_COMMAND_UI(IDR_HERO_LIST, OnUpdateHeroList)
23                                          //退出菜单栏响应函数
24      ON_UPDATE_COMMAND_UI(IDR_EXIT_GAME, OnUpdateExitGame)
25                                          //关于菜单栏响应函数
26      ON_UPDATE_COMMAND_UI(IDR_ABOUT, OnUpdateAbout)
27                                          //开始游戏菜单栏响应函数
28      ON_UPDATE_COMMAND_UI(IDR_START_GAME, OnUpdateStartGame)
29  END_MESSAGE_MAP()
30
31  /////////////////////////////////////////////////////////////////////
32  // CSnakeDlg 中各成员函数的实现
33  BOOL CSnakeDlg::OnInitDialog()
34  {
35      CDialog::OnInitDialog();            //对话框初始化
36
37      CMenu* pSysMenu = GetSystemMenu(FALSE);
38      if (pSysMenu != NULL)
39      {
40          CString strAboutMenu;
41          strAboutMenu.LoadString(IDS_ABOUTBOX);
42          if (!strAboutMenu.IsEmpty())
43          {
44              pSysMenu->AppendMenu(MF_SEPARATOR);
45              pSysMenu->AppendMenu(MF_STRING,IDM_ABOUTBOX, strAboutMenu);
46          }
47      }
48
49
50      SetIcon(m_hIcon, TRUE);                 //设置大图标
51      SetIcon(m_hIcon, FALSE);                //设置小图标
52                                              //加载主菜单
53      m_main_menu.LoadMenu(IDR_MAIN_MENU);
54                                              //设置主菜单
55      SetMenu(&m_main_menu);
56
57      InitMenu();                             //初始化菜单
58
59      return TRUE;                            //返回真，初始化成功
60  }
61
```

```
62  void CSnakeDlg::InitMenu()
63  {//初始化菜单
64      m_main_menu.CheckMenuItem(IDR_LOW, MF_BYCOMMAND | MF_CHECKED);
65      m_main_menu.CheckMenuItem(IDR_HIGH, MF_BYCOMMAND | MF_UNCHECKED);
66      m_main_menu.CheckMenuItem(IDR_NOR, MF_BYCOMMAND | MF_UNCHECKED);
67      m_main_menu.CheckMenuItem(IDR_PLAY_MUSIC,
68                      MF_BYCOMMAND| MF_UNCHECKED);
69  }
70  //播放音乐，菜单栏响应函数
71  void CSnakeDlg::OnUpdatePlayMusic(CCmdUI* pCmdUI)
72  {//判断播放音乐菜单当前状态
73      BOOL bCheck = (BOOL)m_main_menu.GetMenuState(IDR_PLAY_MUSIC,
74                      MF_CHECKED);
75
76      if(bCheck)
77      {
78          m_main_menu.CheckMenuItem(IDR_PLAY_MUSIC,
79                      MF_BYCOMMAND | MF_UNCHECKED);
80      }
81      else
82      {
83          m_main_menu.CheckMenuItem(IDR_PLAY_MUSIC,
84                      MF_BYCOMMAND | MF_CHECKED);
85      }
86      PlayBackMusic(!bCheck);            //调用播放背景音乐功能函数
87  }
88  //“游戏等级”|“高”菜单栏响应函数
89  void CSnakeDlg::OnUpdateHigh(CCmdUI* pCmdUI)
90  {//判断当前菜单状态
91      BOOL bCheck = (BOOL)m_main_menu.GetMenuState(IDR_HIGH, MF_CHECKED);
92
93      if( !bCheck )
94      {
95          m_main_menu.CheckMenuItem(IDR_HIGH,
96                          MF_BYCOMMAND | MF_CHECKED);
97          m_main_menu.CheckMenuItem(IDR_LOW,
98                          MF_BYCOMMAND | MF_UNCHECKED);
99          m_main_menu.CheckMenuItem(IDR_NOR,
100                         MF_BYCOMMAND | MF_UNCHECKED);
101     }
102
103 }
104 //“游戏等级”|“低”菜单栏响应函数
105 void CSnakeDlg::OnUpdateLow(CCmdUI* pCmdUI)
106 {//判断当前菜单状态
107     BOOL bCheck = (BOOL)m_main_menu.GetMenuState(IDR_LOW, MF_CHECKED);
108
109     if( !bCheck )
110     {
111         m_main_menu.CheckMenuItem(IDR_LOW,
112                         MF_BYCOMMAND | MF_CHECKED);
113         m_main_menu.CheckMenuItem(IDR_NOR,
114                         MF_BYCOMMAND | MF_UNCHECKED);
115         m_main_menu.CheckMenuItem(IDR_HIGH,
116                         MF_BYCOMMAND | MF_UNCHECKED);
117     }
118
119 }
120 //“游戏等级”|“中”菜单栏响应函数
```

```
121 void CSnakeDlg::OnUpdateNor(CCmdUI* pCmdUI)
122 {//判断当前菜单状态
123     BOOL bCheck = (BOOL)m_main_menu.GetMenuState(IDR_NOR, MF_CHECKED);
124     if( !bCheck )
125     {
126         m_main_menu.CheckMenuItem(IDR_NOR,
127                                 MF_BYCOMMAND | MF_CHECKED);
128         m_main_menu.CheckMenuItem(IDR_LOW,
129                                 MF_BYCOMMAND | MF_UNCHECKED);
130         m_main_menu.CheckMenuItem(IDR_HIGH,
131                                 MF_BYCOMMAND | MF_UNCHECKED);
132     }
133 }
134 //"游戏帮助"菜单栏响应函数
135 void CSnakeDlg::OnUpdateHelp(CCmdUI* pCmdUI)
136 {
137     Help();                                 //调用帮助功能函数
138 }
139 //"关于"菜单栏响应函数
140 void CSnakeDlg::OnUpdateAbout(CCmdUI* pCmdUI)
141 {
142     CAboutDlg dlgAbout;
143     dlgAbout.DoModal();                     //弹出关于对话框
144 }
145 //"英雄榜"菜单栏响应函数
146 void CSnakeDlg::OnUpdateHeroList(CCmdUI* pCmdUI)
147 {
148     HeroList();                             //调用英雄榜功能函数
149 }
150 //"退出"菜单栏响应函数
151 void CSnakeDlg::OnUpdateExitGame(CCmdUI* pCmdUI)
152 {
153     CDialog::OnCancel();                    //退出对话框功能函数
154 }
155 //"开始游戏"菜单栏响应函数
156 void CSnakeDlg::OnUpdateStartGame(CCmdUI* pCmdUI)
157 {
158     StartGame();                            //调用开始新游戏功能函数
159 }
```

代码解析：通过这段代码，可以得出其中几个主要函数的实现办法如下所述。

- 代码第 71 行“播放背景音乐”菜单栏响应函数的实现是通过调用播放音乐成员函数来实现音乐播放功能。
- 代码第 95、111、126 行“游戏等级”菜单栏响应函数的实现是通过改变当前游戏等级变量的值，来实现游戏等级的选择。
- 代码第 135 行“帮助”菜单栏响应函数的实现是通过创建帮助对话框类的对象，并调用其成员函数，将帮助对话框弹出。
- 代码第 156 行“开始游戏”菜单栏响应函数的实现是通过调用对话框的开始游戏成员函数，来实现开始新游戏功能。

11.5.2　游戏帮助的实现

在贪吃蛇游戏中，帮助功能只需要对游戏方法进行简单的说明。所以在这里，设计一

个对话框用来说明游戏方法即可。

技巧：游戏的帮助一定要简单明了，让玩家一看就知道如何使用。

游戏帮助界面的设计如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 11.8 所示。

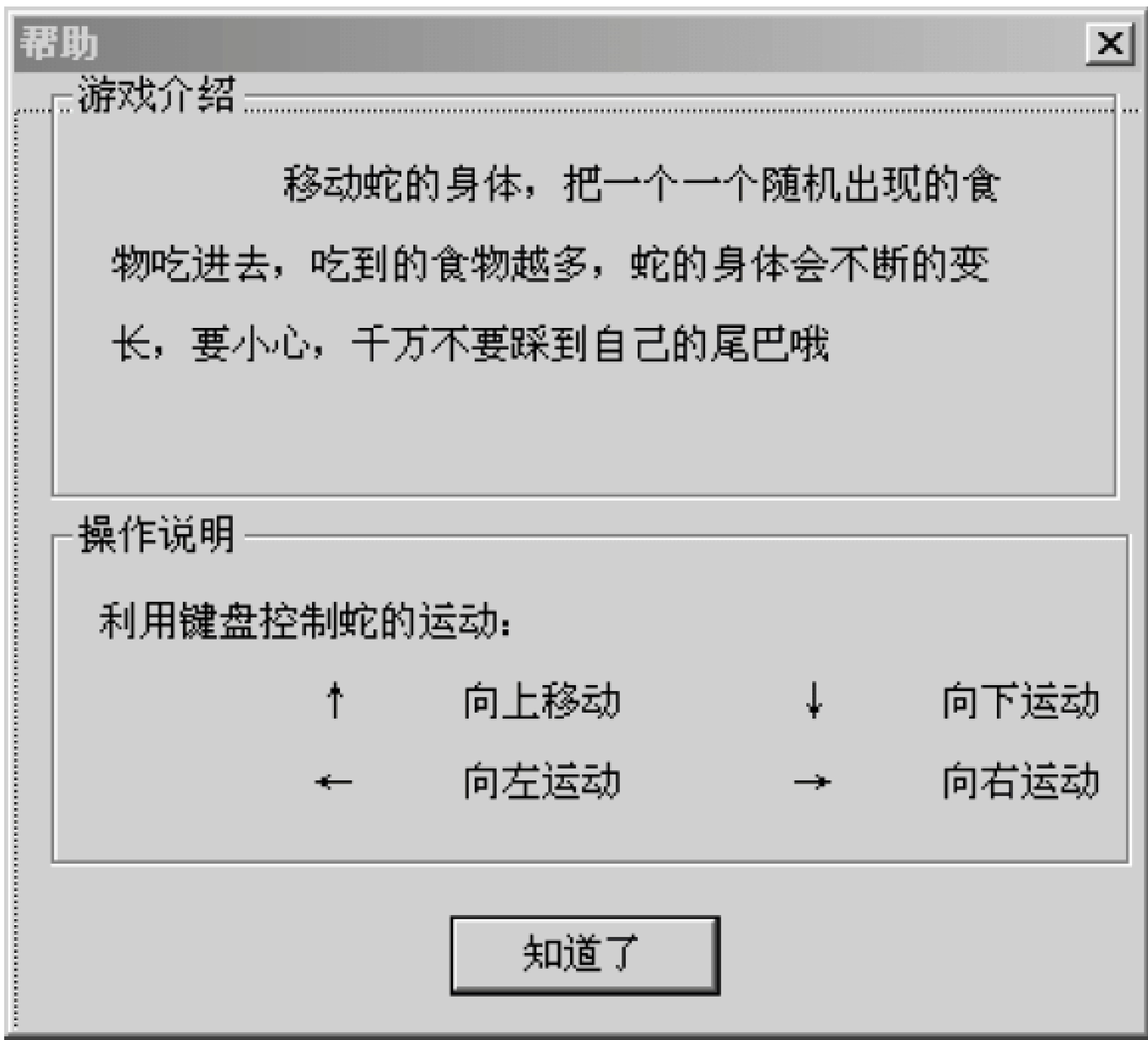


图 11.8　帮助对话框

（2）添加帮助对话框类声明，如代码 11.3 所示。

代码 11.3　帮助对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
04  // HelpDlg.对话框类的头文件
05
06
07  class CHelpDlg : public CDialog
08  {
09  public:
10      CHelpDlg(CWnd* pParent = NULL);         //构造函数
11
12      enum { IDD = IDD_HELP_DLG };            //对话框资源名
13      protected:
14      virtual void DoDataExchange(CDataExchange* pDX);
15
16  protected:
17
18      virtual void OnOK();                    //“知道了”按钮成员函数声明
19      DECLARE_MESSAGE_MAP()
20  };
```

（3）帮助对话框类的实现比较简单，就是基本的对话框类。主要是其中的资源要加载正确。其代码如代码 11.4 所示。

代码 11.4　帮助对话框类的实现

```
01  // HelpDlg.cpp 源文件
02
```

```
03
04  #include "stdafx.h"                          //插入头文件
05  #include "Snake.h"
06  #include "HelpDlg.h"
07
08  ////////////////////////////////////////////////////////////////
09  // CHelpDlg 对话框
10
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/)//构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);        //数据与资源加载
19  }
20  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)     //函数映射表
21
22  END_MESSAGE_MAP()
23
24  ////////////////////////////////////////////////////////////////
25  // CHelpDlg 消息句柄
26
27  void CHelpDlg::OnOK()
28  {
29      CDialog::OnOK();                     //调用 OK 响应函数
30  }
```

（4）实现了帮助对话框类后，还需要实现在 CSnakeDlg 中的 Help()函数。这个函数通过创建 CHelpDlg 类的对象，弹出帮助对话框。其代码如代码 11.5 所示。

代码 11.5　CSnakeDlg 的 Help 函数实现

```
01  #include "HelpDlg.h"                     //注意，要插入类的头文件
02  ...
03  void CSnakeDlg::Help()                   //help 功能函数实现
04  {
05      CHelpDlg dlg;                        //定义 CHelpDlg 对话框类对象
06      dlg.DoModal();                       //弹出对话框
07  }
```

代码解析：整个代码没有太多内容，主要是将帮助对话框显示出来，实现游戏的文字说明功能，如代码第 6 行，弹出帮助对话框并显示。

11.5.3 “英雄榜”的实现

在前面的内容中，已经对“英雄榜”设计方法进行了描述。在这里给出其详细的代码和实现过程。

（1）创建一个英雄榜对话框资源，并加入到工程中。对话框中的资源 ID 及名称如表 11.12 所示。

表 11.12　资源ID及名称对照

ID	名　　称
IDC_NAME_EDIT	姓名编辑框
IDC_SCORE_EDIT	分数编辑框
IDC_LEVEL_EDIT	等级编辑框

（2）对话框的结构如图 11.9 所示。



图 11.9　英雄榜对话框

（3）创建配置文件 Hero.ini，其格式如下：

```
[HERO]
name=XXX
score=0
level=0
```

（4）英雄榜对话框类的声明，如代码 11.6 所示。

代码 11.6　英雄榜对话框类的声明

```
01  #if !defined(AFX_HERODLG_H__)
02  #define AFX_HERODLG_H__
03
04  // HeroDlg.h 头文件
05
06  ////////////////////////////////////////////////////////////////////
07  // CHeroDlg dialog 对话框类声明
08
09  class CHeroDlg : public CDialog
10  {
11  public:
12      void SetWriteFlg(BOOL bflg);       //接口函数，设置可记录标志变量
13      CHeroDlg(CWnd* pParent = NULL);    //构造函数
14
15      enum { IDD = IDD_HERO_LIST };      //对话框资源
16      int     m_level;                   //保存等级变量
17      CString m_name;                    //保存姓名变量
18      int     m_score;                   //保存分数变量
19
20      public:
21      virtual int DoModal();             //弹出对话框函数声明
22      protected:
23      virtual void DoDataExchange(CDataExchange* pDX);
24
25  protected:
```

```
26
27      virtual void OnOK();                   //单击按钮响应函数声明
28      DECLARE_MESSAGE_MAP()
29  private:
30      BOOL m_bWriteflg;                      //记录标志变量
31  };
32
33  #endif
```

（5）“英雄榜”对话框分为两种情况，一种是显示当前最高记录；另一种是写入当前最高记录。所以必须在其内部有一个状态标志位变量来区别。当需要写入时，设置其为有效。需要显示时，设置其为无效。在对话框的实现中，需要对配置文件进行读写操作，如代码 11.7 所示。

技巧： 配置文件的操作可能通过系统 API 函数 GetPrivateProfileString（读取）和 WritePrivateProfileString（写入）来进行。

代码 11.7　英雄榜对话框类的实现

```
01  // HeroDlg.cpp 源文件
02  #include "stdafx.h"                         //插入头文件
03  #include "Snake.h"
04  #include "HeroDlg.h"                        //插入类声明头文件
05
06  ////////////////////////////////////////////////////////////////////
07  // CHeroDlg 对话框
08
09  CHeroDlg::CHeroDlg(CWnd* pParent /*=NULL*/)//构造函数
10      : CDialog(CHeroDlg::IDD, pParent)
11  {
12      m_bWriteflg = FALSE;                    //初始化写标志为假
13  }
14
15  void CHeroDlg::DoDataExchange(CDataExchange* pDX)
16  {
17      CDialog::DoDataExchange(pDX);           //变量与资源映射
18      //{{AFX_DATA_MAP(CHeroDlg)
19      DDX_Text(pDX, IDC_LEVEL_EDIT, m_level);
20      DDX_Text(pDX, IDC_NAME_EDIT, m_name);
21      DDX_Text(pDX, IDC_SCORE_EDIT, m_score);
22      DDV_MinMaxInt(pDX, m_score, 0, 10000); //设置最大和最小可输入值
23      //}}AFX_DATA_MAP
24  }
25
26
27  BEGIN_MESSAGE_MAP(CHeroDlg, CDialog)
28      ON_BN_CLICKED(IDOK_BTN, OnBtn)          //按钮与函数映射
29  END_MESSAGE_MAP()
30
31  ///////////////////////////////////////////////////////////////////////
32  // CHeroDlg 消息句柄
33
34  void CHeroDlg::SetWriteFlg(BOOL bflg)       //设置写入标志
35  {
36      m_bWriteflg = bflg;
37  }
```

```
38
39  int CHeroDlg::DoModal()                             //弹出对话框
40  {
41      char pszTmp[128] = {0};
42
43      //读取配置文件
44      GetPrivateProfileString("HERO", "name", "0",
45          pszTmp, 127, ".\\hero.ini");                //读取姓名
46      m_name = CString(pszTmp);
47
48      if(!m_bWriteflg)
49      {
50          GetPrivateProfileString("HERO", "score", "0",
51              pszTmp, 127, ".\\hero.ini");            //读取分数
52          m_score = atoi(pszTmp);
53          GetPrivateProfileString("HERO", "level", "0",
54              pszTmp, 127, ".\\hero.ini");            //读取等级
55          m_level = atoi(pszTmp);
56      }
57
58      return CDialog::DoModal();
59  }
60
61  void CHeroDlg::OnBtn()                              //按钮响应
62  {
63      UpdateData(TRUE);                               //更新显示数值到变量
64      if(m_bWriteflg)
65      {
66          CString tmp;
67          tmp.Format("%d", m_score);                  //格式化数值为字符串
68          WritePrivateProfileString("HERO", "name", m_name, ".\\hero.ini");
69          WritePrivateProfileString("HERO", "score", tmp, ".\\hero.ini");
70          tmp.Format("%d", m_level);
71          WritePrivateProfileString("HERO", "level", tmp, ".\\hero.ini");
72      }
73      m_bWriteflg = FALSE;
74
75      CDialog::OnOK();
76  }
77
78  BOOL CHeroDlg::OnInitDialog()                       //初始化对话框
79  {
80      CDialog::OnInitDialog();
81
82      if(m_bWriteflg)
83      {                                               //当为写入时，改变按钮名称
84          SetDlgItemText(IDOK_BTN, "记录");
85      }
86
87      return TRUE;
88  }
```

代码解析：第 44 行是通过调用 GetPrivateProfileString()函数来读取配置文件中相关数据。代码第 68～71 行，是调用 WritePrivateProfileString()函数将指定的数据写入到配置文件中。

（6）CSnakeDlg 中的 HeroList()函数主要创建 CHeroDlg 类对象，再通过该对象来调用弹出对话框函数，如代码 11.8 所示。

代码 11.8　CSnakeDlg 中的 HeroList 函数

```
01  #include "HeroDlg.h"
02  ...
03  void CSnakeDlg::HeroList()
04  {
05      CHeroDlg dlg;                   //英雄对话框类对象
06      dlg.DoModal();                  //弹出对话框
07  }
```

11.5.4　游戏背景音乐播放的实现

游戏背景音乐播放，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。要实现这个功能，需要如下几个步骤。

（1）在工程文件中，添加 winmm.lib 静态库文件及头文件，参见 5.4 节。

（2）实现 CSnakeDlg 中的 PlayBackMusic()函数，如代码 11.9 所示。

代码 11.9　CSnakeDlg 中的 PlayBackMusic 函数实现

```
01  #include <mmsystem.h>                   //插入 API 头文件
    ...
02  void CSnakeDlg::PlayBackMusic(BOOL bflg)
03  {
04      //指定文件并播放
05      if(bflg)
06      {                                   //播放音乐
07          sndPlaySound("music.wav",SND_ASYNC);
08      }
09      else
10      {                                   //停止播放
11          sndPlaySound(NULL,SND_PURGE);
12      }
13  }
```

代码解析：第 7 行，调用系统 API 函数 sndPlaysound()来实现指定 wav 音频文件的播放，参数 SND_ASYNC 是播放，SND_PURGE 是停止。

11.6　贪吃蛇游戏核心算法的设计与实现

前面已经对游戏界面上的设计和实现进行了讲解。本节将主要讲解贪吃蛇游戏中核心类算法的设计与实现。

11.6.1　主游戏类的设计

主游戏类主要负责贪吃蛇及果实的显示和更新。

1. 果实出现的设计思路

（1）采用随机数生成果实出现坐标。
（2）判断当前生成的果实坐标是否在贪吃蛇身体范围内。
（3）如果在，重新生成直到不在为止。如果不在，则把坐标位置返回给调用对象。

2. 贪吃蛇更新的设计思路

（1）接收玩家按下的方向键消息，并保存到方向变量中。
（2）定义一个时间定时器，时间间隔由游戏等级决定，如表 11.13 所示。

表 11.13　时间间隔与游戏等级对照

时 间 间 隔	游 戏 等 级
100ms	低
50ms	中
30ms	高

（3）当每次时间间隔到达时，则根据方向变量来更新贪吃蛇 BODY 向量。

（4）判断 BODY 向量的第一个元素中的坐标数据是否碰到边界或者蛇身，如果有，转到第（7）步。

（5）判断 BODY 向量的第一个元素中的坐标数据是否与当前果实坐标重合，如果有，表示贪吃蛇已经吃到果实。这时就向贪吃蛇 BODY 向量添加一个元素，并重新生成一个果实。

（6）重绘整个贪吃蛇界面及果实。重复前面步骤（1）～（6）。

（7）游戏结束时，计算当前游戏得分。如果超过最高分，设置英雄榜中的写入标志并弹出英雄榜对话框；否则提示游戏结束。

11.6.2　主游戏类的实现

主游戏类的声明中包含了绘制蛇身函数、初始化游戏函数、随机分配果实函数及设置当前游戏等级函数等的声明。其代码如代码 11.10 所示。

代码 11.10　主游戏类的声明

```
01  #ifndef  __SNAKE_GAME_H__
02  #define  __SNAKE_GAME_H__
03  //主游戏类
04
05  #include "Afxtempl.h"                       //插入模板头文件
06                                              //定义各种宏
07  #define GAME_LEVEL_LOW    2
08  #define GAME_LEVEL_NOR    4
09  #define GAME_LEVEL_HIGH   8
10
11  #define DIREC_UP          1                 //向上标志
12  #define DIREC_DOWN        2                 //向下标志
13  #define DIREC_RIGHT       3                 //向右标志
```

```
14  #define DIREC_LEFT        4                //向左标志
15
16  #define LOW_LEVEL_SLEEP   100
17  #define NOR_LEVEL_SLEEP   50
18  #define HIGH_LEVEL_SLEEP  30
19
20  class CSnakeGame:public CWnd             //主游戏类的声明
21  {
22  public:
23      void HeroWrite();                    //弹出英雄榜，并写入
24      void ReDrawBody(CPoint pt);          //重绘一个点
25      void InitGame();                     //初始化游戏
26      CSnakeGame();                        //构造函数
27      virtual ~CSnakeGame();               //析构函数
28
29      BOOL GameStart();                    //游戏开始函数
30      void InitFoods();                    //初始化果实函数
31      void SetGameLevel(int level);        //设置游戏等级
32
33  private:
34
35      CPoint  m_psFood;                    //当前食物坐标
36      int     m_nDirect;                   //当前蛇先进方向
37      int     m_nScore;                    //当前游戏分数
38      int     m_nlevel;                    //当前游戏等级
39      int    m_nHighScore;                 //当前游戏最高分
40
41      CArray<CPoint,CPoint> m_body;        //蛇身向量
42  protected:
43     afx_msg void OnPaint();
44      afx_msg void OnTimer(UINT nIDEvent);
45      afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
46     DECLARE_MESSAGE_MAP()
47  };
48
49  #endif
```

通过前面类的声明，已经知道主游戏类中包含了的几个基本函数，如构造、析构、绘图和接收键盘输入等。这些函数是主游戏窗口类基础，用于构成对话框、处理图片的显示及人机交互过程。其代码如代码 11.11 所示。

代码 11.11　主游戏类中的功能函数实现

```
01  CSnakeGame::CSnakeGame()
02  {
03      m_nScore = 0;                    //初始化分数
04      m_psFood.x = 30;                 //初始化贪吃蛇 X 坐标
05      m_psFood.y = 30;                 //初始化贪吃蛇 Y 坐标
06      m_nHighScore = 0;                //初始化最高分
07  }
08
09  CSnakeGame::~CSnakeGame()            //析构函数
10  {
11  }
12
13  // 消息映射表
14  BEGIN_MESSAGE_MAP( CSnakeGame, CWnd )
```

```
15  ON_WM_PAINT()
16  ON_WM_TIMER()                          //时间响应函数
17  ON_WM_KEYUP()                          //按键释放响应函数
18  END_MESSAGE_MAP()
19  // 处理 WM_PAINT 消息
20  void CSnakeGame::OnPaint()
21  {
22      CPaintDC dc( this );
23      CDC MemDC;
24      MemDC.CreateCompatibleDC( &dc );//创建内存绘制 DC 对象
25      // 装载背景
26      CBitmap bmp;
27      CPen pen;
28      bmp.LoadBitmap( IDB_BMP_BJ );      //加载图片
29      pen.CreatePen( PS_SOLID, 1, 0xff );
30      MemDC.SelectObject( &bmp );
31      MemDC.SelectObject( &pen );
32      MemDC.SetROP2( R2_NOTXORPEN );
33       CString ysStr;                    //定义字符串用于显示游戏时间、得分等级
34                                         //设置字体背景
35       MemDC.SetBkMode(TRANSPARENT);
36       MemDC.SetTextColor(67);           //设置字体颜色并初始化字符串
37       ysStr.Format("当前得分:%d",m_nScore);
38       MemDC.TextOut(30,50,ysStr);       //输出文本
39       switch(m_nlevel)                  //根据等级判断
40       {
41       case GAME_LEVEL_LOW:              //如果是初级
42           ysStr.Format("当前等级:初级");
43           break;
44       case GAME_LEVEL_NOR:              //如果是中级
45           ysStr.Format("当前等级:中级");
46           break;
47       case GAME_LEVEL_HIGH:             //如果是高级
48           ysStr.Format("当前等级:高级");
49           break;
50       }
51       MemDC.TextOut(30,30,ysStr);       //输出字符
52
53                                         //绘制蛇的样式
54       CPen yspen;
55                                         //定义白色画笔绘制蛇的边框
56       yspen.CreatePen(1,1,RGB(255,255,255));
57       MemDC.SelectObject(&yspen);
58       CBrush ysbrush;
59       //
60       int k=m_body.GetUpperBound()+2;//设置一个变量存储贪吃蛇的身体长度
61       if(k<=10)                         //如果小于 10，那么就为绿色
62       {
63           ysbrush.CreateSolidBrush(RGB(0,255,0));
64           MemDC.SelectObject(&ysbrush);
65           //绘制果实
66           MemDC.Rectangle(
67               CRect(10+m_psFood.y*10,
68               120+m_psFood.x*10,
69               10+(m_psFood.y+1)*10,
70               120+(m_psFood.x+1)*10)
71               );
```

```
72        }
73        else if(k>10&&k<=20)              //如果在 10 和 20 之间，那么就为绿色
74        {
75            ysbrush.CreateSolidBrush(RGB(0,0,255));
76            MemDC.SelectObject(&ysbrush);
77            //绘制果实
78            MemDC.Rectangle(
79                CRect(10+m_psFood.y*10,
80                120+m_psFood.x*10,
81                10+(m_psFood.y+1)*10,
82                120+(m_psFood.x+1)*10)
83                );
84        }
85        else if(k>20&&k<=30)              //如果在 20 和 30 之间，那么就为绿色
86        {
87            ysbrush.CreateSolidBrush(RGB(255,255,0));
88            MemDC.SelectObject(&ysbrush);
89            //绘制果实
90            MemDC.Rectangle(
91                CRect(10+m_psFood.y*10,
92                120+m_psFood.x*10,
93                10+(m_psFood.y+1)*10,
94                120+(m_psFood.x+1)*10)
95                );
96        }
97        else                              //其余情况均为红色
98        {
99            ysbrush.CreateSolidBrush(RGB(255,0,0));
100           MemDC.SelectObject(&ysbrush);
101                                         //绘制果实
102           MemDC.Rectangle(
103               CRect(10+m_psFood.y*10,
104               120+m_psFood.x*10,
105               10+(m_psFood.y+1)*10,
106               120+(m_psFood.x+1)*10)
107               );
108       }
109                                         //初始化点数组
110       for(int i=0;i<=m_body.GetUpperBound();i++)
111       {
112           CPoint ysPoint=m_body.GetAt(i);
113          MemDC.Rectangle(
114               CRect(10+ysPoint.y*10,
115               120+ysPoint.x*10,
116               10+(ysPoint.y+1)*10,
117               120+(ysPoint.x+1)*10)
118               );
119       }
120       dc.BitBlt( 0, 0, 325, 425, &MemDC,0, 0, SRCCOPY );
121 }
122 // 处理按键消息
123 void CSnakeGame::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
124 {
125       switch(nChar)
126       {
127       case VK_UP:                       //按向上键
128           m_nDirect=DIREC_UP;           //控制方向变量为 DIREC_UP
129           break;
```

```
130     case VK_DOWN:                    //按向下键
131         m_nDirect=DIREC_DOWN;        //控制方向变量为 DIREC_DOWN
132         break;
133     case VK_LEFT:                    //按向左键
134         m_nDirect=DIREC_LEFT;        //控制方向变量为 DIREC_LEFT
135         break;
136     case VK_RIGHT:                   //按向右键
137         m_nDirect=DIREC_RIGHT;       //控制方向变量为 DIREC_RIGHT
138         break;
139     default:
140         break;
141     }
142 }
```

代码解析：代码第 66 行，通过得到果实对象的坐标，绘制果实对象到屏幕上。但要注意，必须将果实绘制到主界面内，即 X 大于 10，Y 大于 120 的坐标外。代码第 110 行，通过循环语句遍历蛇身数组实现贪吃蛇的显示。代码第 123 行，通过响应键盘按下信息的响应，实现贪吃蛇方向的方向。

除了上面讲解的这些基本函数外，主游戏窗口类中，还应该包含其他功能函数。如初始化果实函数、初始化游戏、设置游戏等级及定时器处理。游戏中通过调用这些功能函数，来实现全部的游戏功能，如代码 11.12 所示。

代码 11.12　主游戏类中的功能函数实现

```
01  // 初始化果实
02  void CSnakeGame::InitFoods()
03  {
04      int m_ysX,m_ysY;
05      while(1)
06      {
07          m_ysX=rand()%28;         //随机生成横坐标使其与贪吃蛇的身体可以接上
08          m_ysY=rand()%28;         //随机生成纵坐标使其与贪吃蛇的身体可以接上
09          for(int i=0;i<=m_body.GetUpperBound();i++)
10          {
11                                   //获取贪吃蛇的身体坐标
12              CPoint ysPoint1=m_body.GetAt(i);
13                                   //如果身体的横坐标或纵坐标与果实的横纵坐标相同
14              if(ysPoint1.x!=m_ysX||ysPoint1.y!=m_ysY)
15              {
16                                   //将随机出现的坐标记录为果实坐标
17                  m_psFood = CPoint(m_ysX, m_ysY);
18                  return;
19              }
20          }
21      }
22  }
23  // 设置游戏等级
24  void CSnakeGame::SetGameLevel(int level)
25  {
26      m_nlevel = level;
27  }
28  // 开始游戏函数
29  BOOL CSnakeGame::GameStart()
30  {
31      int nSleep = LOW_LEVEL_SLEEP;         //设置游戏默认时间间隔
32
```

```
33      char pszTmp[128] = {0};
34
35      switch(m_nlevel)
36      {
37      case GAME_LEVEL_HIGH:                 //高级
38          nSleep = HIGH_LEVEL_SLEEP;
39          break;
40      case GAME_LEVEL_NOR:                  //中级
41          nSleep = NOR_LEVEL_SLEEP;
42          break;
43      }
44
45      GetPrivateProfileString("HERO", "score", "0",
46          pszTmp, 127, ".\\hero.ini");      //读取游戏记录分数
47      m_nHighScore = atoi(pszTmp);
48
49      SetTimer(1, nSleep, NULL);            //设置定时器
50
51      InitGame();                           //初始化游戏
52
53      return TRUE;
54  }
55  // 定时器响应函数
56  void CSnakeGame::OnTimer(UINT nIDEvent)
57  {
58      CPoint ysPoint=m_body.GetAt(0);       //获取蛇身的第一个点坐标
59      BOOL bTag = FALSE;                    //定义判断死亡的变量
60      CRule rule;                           //游戏规则类
61
62      switch(m_nDirect)                     //根据键盘按下键来选择蛇移动的方向
63      {
64      case DIREC_DOWN:                      //方向变量向下
65          ysPoint.x++;                      //点纵坐标自加
66          break;
67      case DIREC_UP:                        //方向变量向上
68          ysPoint.x--;                      //点纵坐标自减
69          break;
70      case DIREC_RIGHT:                     //方向变量向右
71          ysPoint.y++;                      //点横坐标自加
72          break;
73      case DIREC_LEFT:                      //方向变量向左
74          ysPoint.y--;                      //点横坐标自减
75          break;
76      }
77
78      if(rule.IsOver(ysPoint, m_body))      //游戏结束
79      {
80          KillTimer(1);                     //关闭定时器
81
82          if(m_nScore > m_nHighScore)       //超过最高分
83          {
84              HeroWrite();                  //写入英雄榜
85          }
86          else
87          {
88              AfxMessageBox("游戏结束,你的分数太低了,没有能进入英雄榜");
89          }
90      }
```

```
91        else
92        {
93            m_body.InsertAt(0,ysPoint)  ;     //将新点添加到蛇的身体中
94            ReDrawBody(ysPoint);              //重绘蛇的身体
95            if(ysPoint.x==m_psFood.x&&ysPoint.y==m_psFood.y)
96            { //如果蛇的身体与果实坐标重合
97                                              //获取蛇身体的长度
98                int nlen=m_body.GetUpperBound();
99                                              //统计分数
100               m_nScore = m_nlevel * (nlen-3);
101
102               InitFoods();                  //再出现下一个果实
103
104               Invalidate();                 //窗口重绘
105           }
106           else
107           {//将最后一个赋给 pt，并移出最后一个，重新绘制 pt 这个点
108               CPoint pt=m_body.GetAt(m_body.GetUpperBound());
109               m_body.RemoveAt(m_body.GetUpperBound());
110               ReDrawBody(pt);               //重新绘制 pt 这点
111           }
112       }
113 }
114 // 初始化游戏
115 void CSnakeGame::InitGame()
116 {
117       m_body.RemoveAll();                   //移出全部数据
118
119       m_body.Add(CPoint(3, 8));             //增加蛇身数据
120       m_body.Add(CPoint(3, 7));
121       m_body.Add(CPoint(3, 6));
122       m_body.Add(CPoint(3, 5));
123
124       srand((unsigned)time(NULL));          //初始化随机数生成器
125
126       m_nDirect = DIREC_RIGHT;              //初始时向右
127       m_nScore  = 0;                        //初始化分数
128
129       InitFoods();                          //初始化果实函数
130
131       Invalidate();                         //重绘窗口
132 }
133 //重绘指定点
134 void CSnakeGame::ReDrawBody(CPoint pt)
135 {
136       InvalidateRect(
137           CRect(10+pt.y*10,
138           120+pt.x*10,
139           10+(pt.y+1)*10,
140           120+(pt.x+1)*10)
141           );
142 }
143 // 英雄榜写入及弹出
144 void CSnakeGame::HeroWrite()
145 {
146       CHeroDlg dlg;
147       dlg.SetWriteFlg(TRUE);            //设置可标志
148       dlg.m_level = m_nlevel;           //设置等级
```

```
149     dlg.m_score = m_nScore;         //设置分数
150     dlg.DoModal();                  //弹出对话框
151 }
```

代码解析：代码第 7 行，在初始化函数中，调用 rand()函数生成随机坐标，然后再用这个随机数取模 28，这样就保证生成的果实数据必须在显示界面内。

11.6.3　游戏规则类的设计

游戏规则类，主要负责游戏规则的处理。在贪吃蛇游戏中，主要有如下两种规则需要实现。

（1）当贪吃蛇碰到四周的边界时，认定贪吃蛇死亡，游戏结束。

（2）当贪吃蛇碰到自己的身体时，认定贪吃蛇死亡，游戏结束。

要实现这两个规则，只需要对贪吃蛇的行走路径进行判断。每走一步时，对当前蛇头位置进行判断，如果蛇头的坐标超出边界或者与贪吃蛇 BODY 向量中的某个坐标数据相同时，就认定贪吃蛇死亡。

11.6.4　游戏规则类的实现

游戏规则类的声明中，主要包含一个成员函数，即判断当前贪吃蛇是否已经死亡的接口函数，其代码如代码 11.13 所示。

代码 11.13　游戏规则类的声明

```
01  //rule.h  游戏规则类的头文件
02  #ifndef __RULE_H__
03  #define __RULE_H__
04
05  #include <afxtempl.h>
06
07  #define MAX_POINT 29                        //最大边界值
08  #define MIN_POINT 0                         //最小边界值
09
10  class CRule                                 //游戏规则类
11  {
12  public:
13                                              //判断当前点是否为游戏结束
14      BOOL IsOver(CPoint &pt, CArray<CPoint, CPoint> &body);
15
16  };
17
18  #endif
```

在游戏规则类的游戏结束判断接口函数中，通过如下办法实现：

先对当前蛇身数组进行遍历。判断如果其中有一个纵坐标多于最下端边框或者少于最上端边框，则说明游戏结束。再判断如果其中有一个横坐标多于最右端边框或者最左端边框，也说明游戏结束。如果都不是，就直接返回不是游戏结束。其代码如代码 11.14 所示。

代码 11.14　游戏规则类的实现

```
01  //rule.cpp 游戏规则类的源文件
```

```
02  #include "stdafx.h"
03  #include "Rule.h"
04
05  BOOL CRule::IsOver(CPoint &pt, CArray<CPoint, CPoint> &body)
06  {   //如果纵坐标多于最下端边框或者少于最上端边框
07      if(pt.x>=MAX_POINT || pt.x<MIN_POINT)
08      {
09          return TRUE;                    //判断死亡
10      }
11      //横坐标多于最右端边框或者最左边边框
12      else if(pt.y>=MAX_POINT29 || pt.y<MIN_POINT)
13      {
14          return TRUE;                    //判断死亡
15      }
16
17      for(int i=0;i<=body.GetUpperBound();i++)
18      {
19          CPoint pt1 = body.GetAt(i);     //将蛇身体的坐标传递给新定义的点
20          if(pt.x==pt1.x&&pt.y==pt1.y)    //如果两点完全相同，说明蛇碰到身体
21          {
22              return TRUE;                //判断死亡
23          }
24      }
25
26      return FALSE;                       //返回假，表示未死亡
27  }
```

代码解析：代码第 17～24 行，是利用循环语句遍历整个蛇身数组，只要在蛇身数组中有一个点与坐标上的点相同，则说明蛇碰到身体，游戏结束。

11.7　贪吃蛇游戏的整合测试

本节主要讲解根据测试用例文档来进行贪吃蛇游戏的整合测试。整个测试用例一共有 9 种。在这里，笔者只对其中几个主要的测试用例进行演示。未演示的测试用例请读者自己去试一下。

11.7.1　游戏等级测试的演示

这里只演示选中“高”等级的情况。根据测试用例文档，其测试步骤如下所述。

（1）选中“游戏设置”|“游戏等级”菜单项。在弹出的下级菜单中，选中“高”等级，如图 11.10 所示。

（2）查看在游戏主界面上，显示的游戏等级是否与选择的一致，如图 11.11 所示。



图 11.10　选择“高”等级

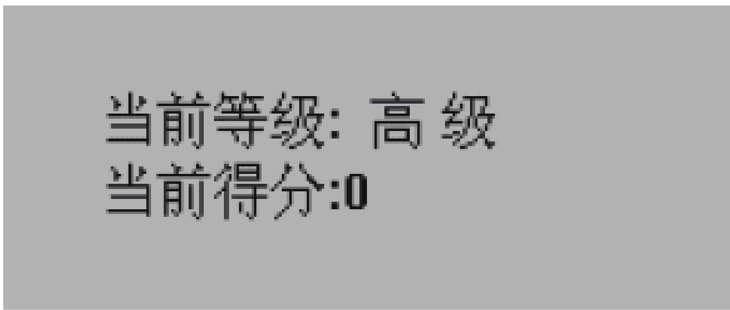


图 11.11　游戏界面上显示的游戏等级

判断结果：游戏等级设置成功。填写测试用例编号 1.7.1 结果为“通过”。

11.7.2　游戏主界面显示功能测试的演示

游戏主界面显示功能的测试，主要是为了测试是否正确显示背景、贪吃蛇及果实。根据测试用例文档，测试步骤如下所述。

（1）直接运行贪吃蛇游戏，并已经开始游戏，如图 11.12 所示。

（2）查看游戏主界面，是否正确显示背景、贪吃蛇及果实，如图 11.13 所示。



图 11.12　选择“开始游戏”选项

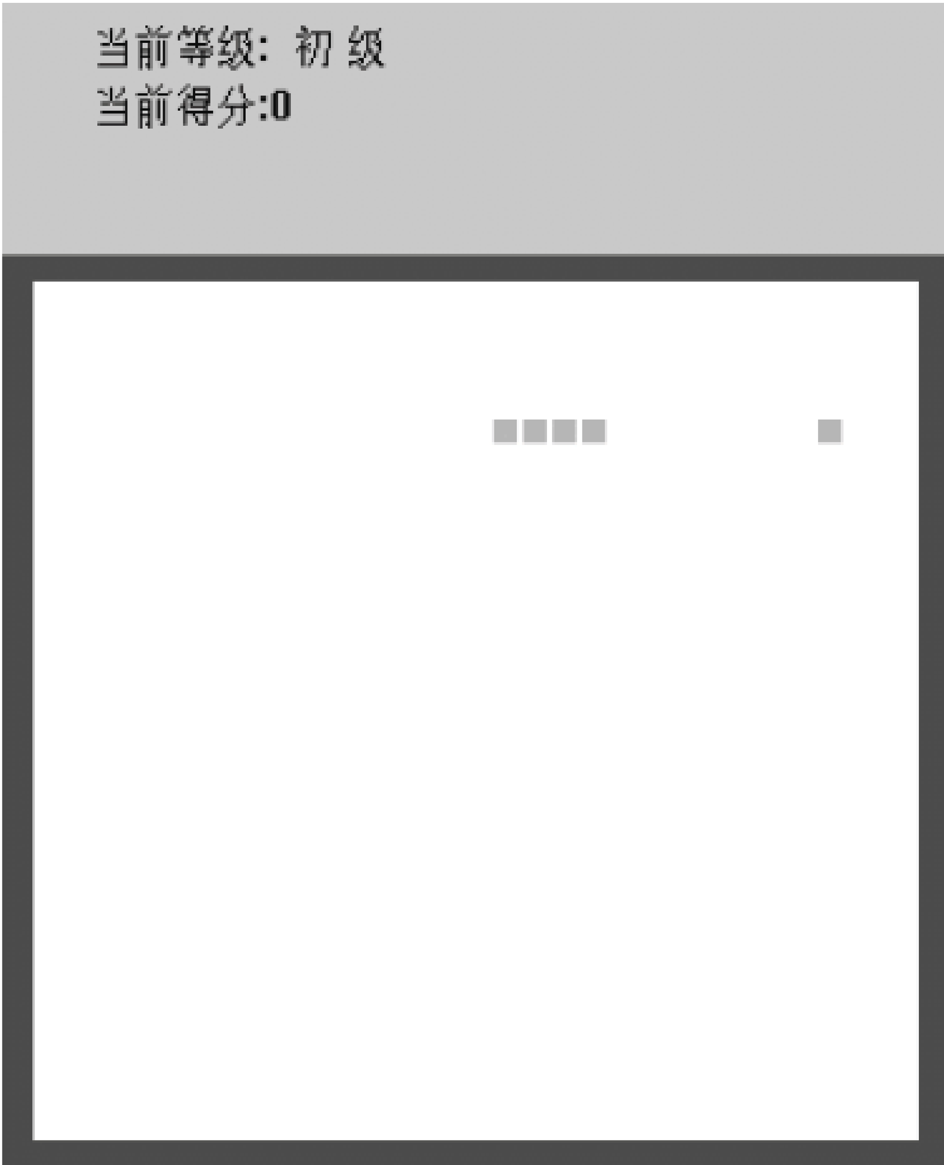


图 11.13　游戏主界面

判断结果：游戏主界面显示功能正确。填写测试用例编号 1.7.9 结果为“通过”。

11.7.3　贪吃蛇移动功能测试的演示

测试游戏中的贪吃蛇是否能够按指定方向移动，这里笔者只演示对其中“下”和“左”方向的测试。根据测试用例文档，其测试步骤如下所述。

（1）开始游戏后，单击“下”按键，控制移动贪吃蛇向下移动，查看是否移动成功，如图 11.14 所示。

（2）向下移动后，单击“左”按键，控制贪吃蛇向左移动，查看是否移动成功，如图 11.15 所示。



图 11.14　贪吃蛇向下移动

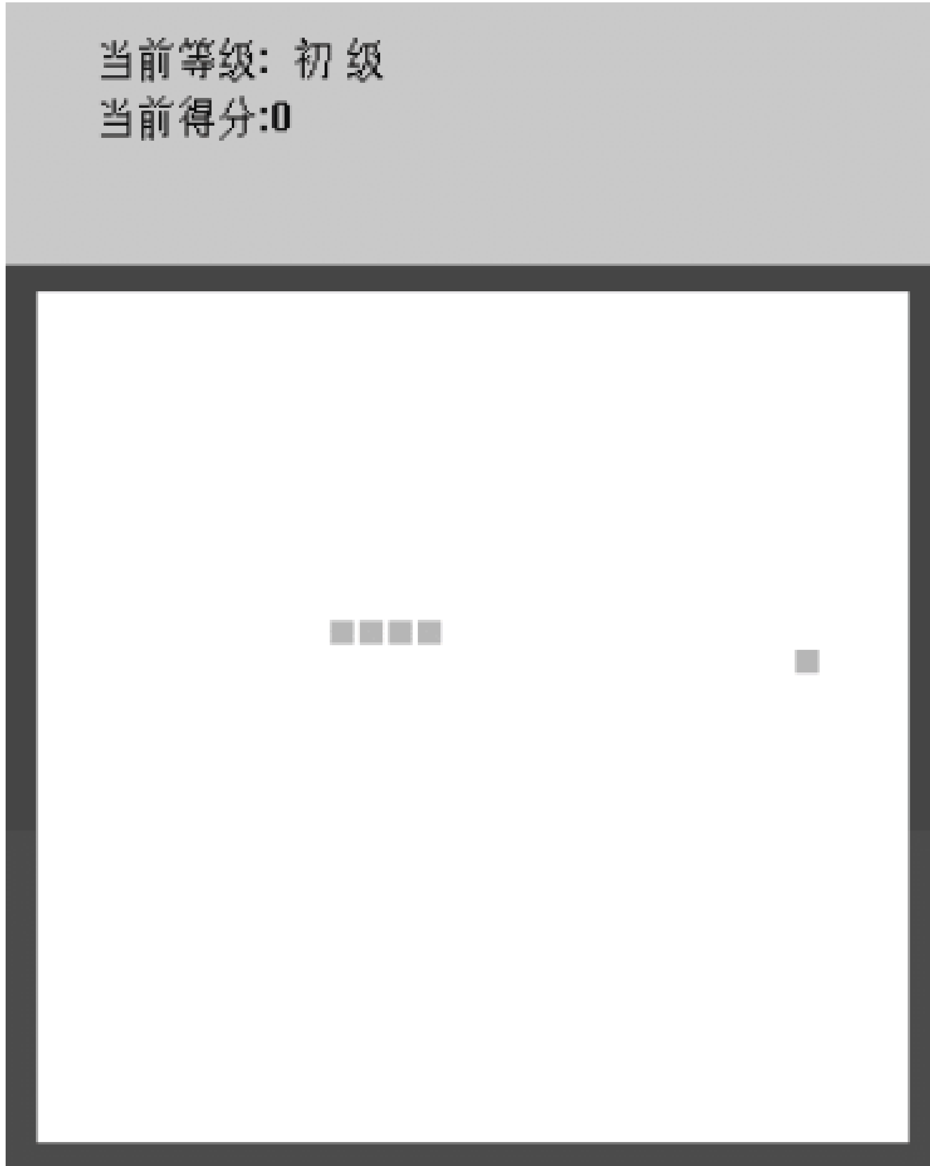


图 11.15　贪吃蛇向左移动

判断结果：贪吃蛇移动功能正确。填写测试用例编号 1.7.2 结果为“通过”。

11.7.4　贪吃蛇游戏规则测试的演示

测试游戏中的贪吃蛇碰到边界或者蛇身时，是否会判定为死亡。根据测试用例文档，其游戏步骤如下所述。

（1）开始游戏后，让贪吃蛇吃到几点果实后，移动到边界上，查看是否会有死亡提示或者英雄榜记录对话框。如图 11.16 所示。

（2）开始游戏后，让贪吃蛇吃到几点果实后，碰到自己的身体，查看是否会有死亡提示或者英雄榜记录对话框出现。如图 11.17 所示。

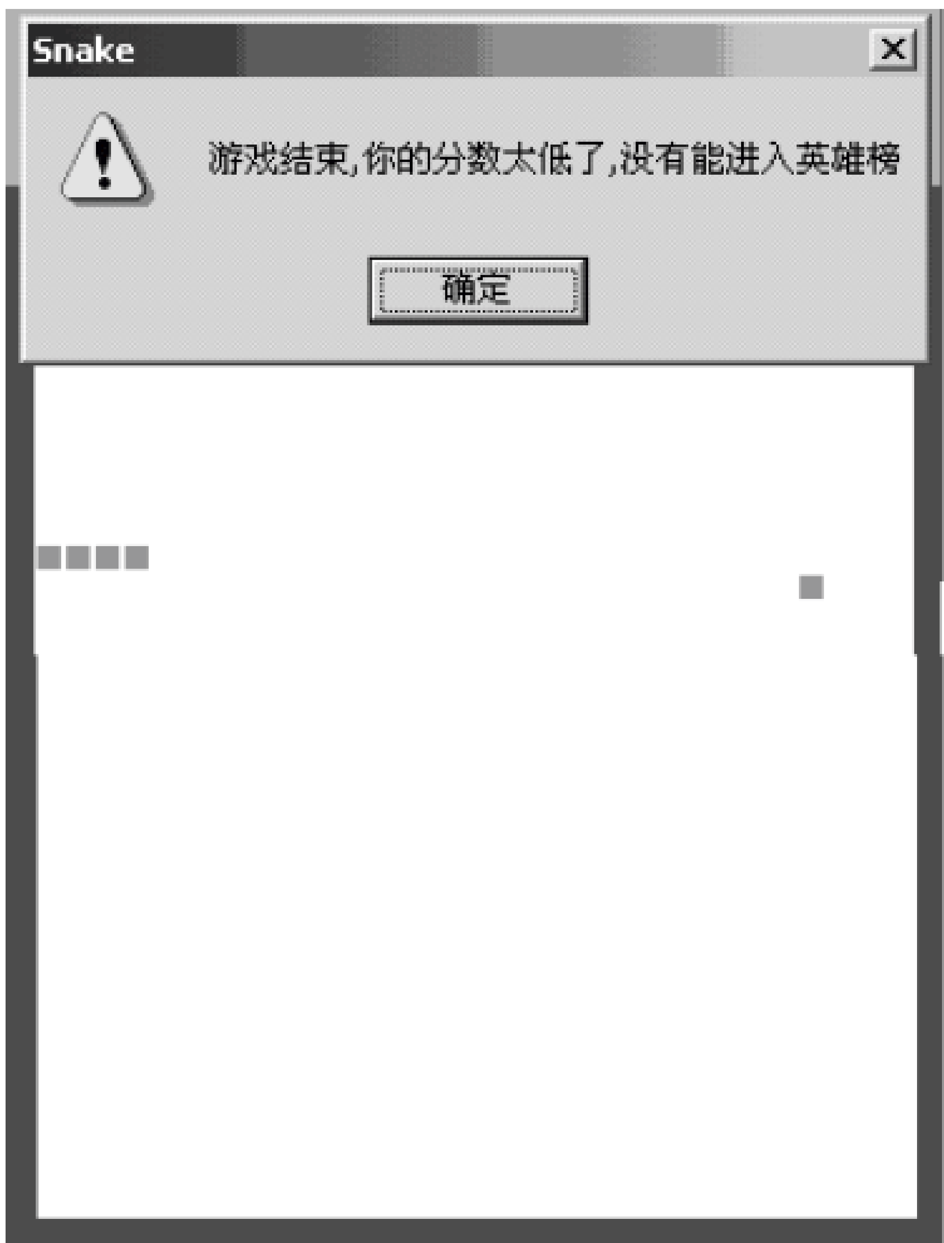


图 11.16　碰到边界死亡

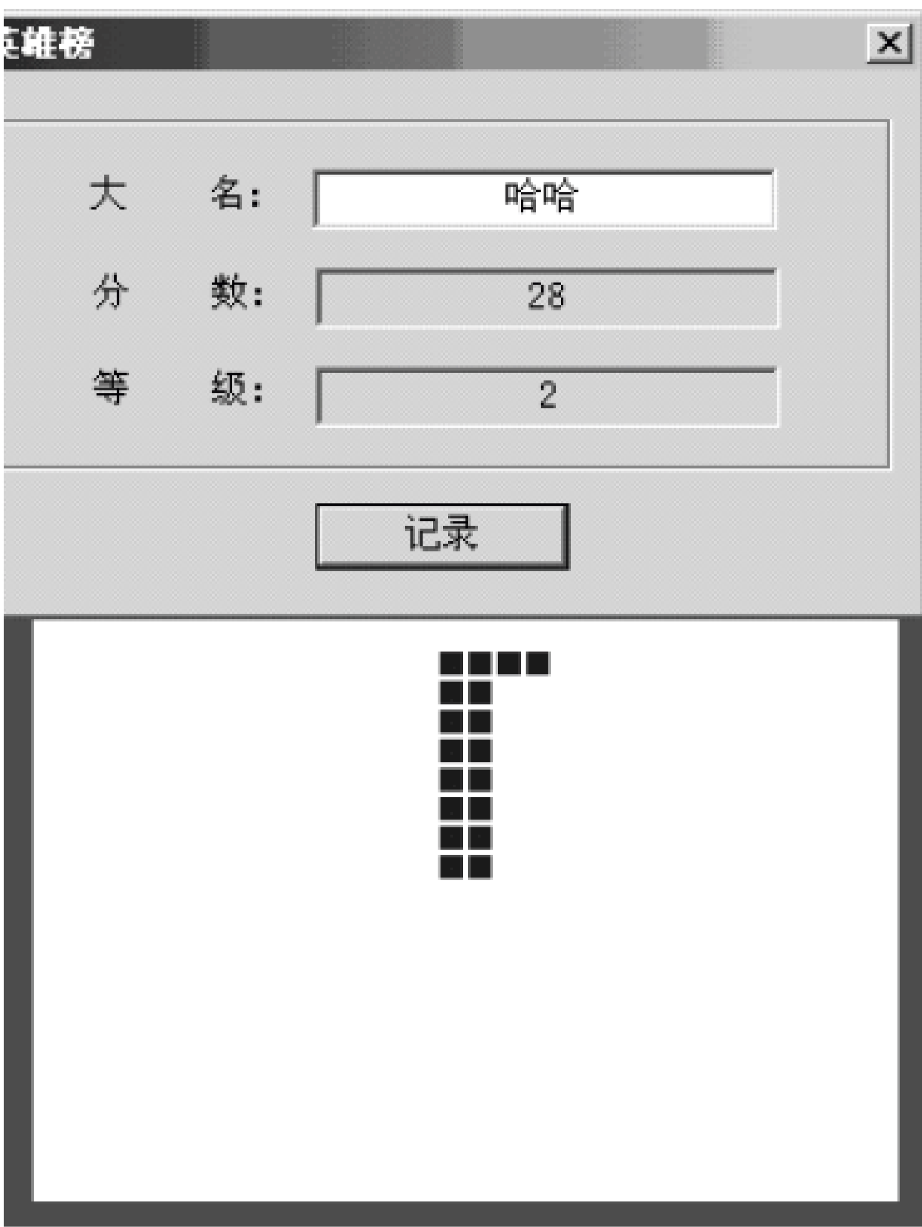


图 11.17　碰到身体死亡

判断结果：贪吃蛇移动功能正确。填写测试用例编号 1.7.5 结果为“通过”。

11.7.5　贪吃蛇游戏分数统计测试的演示

测试游戏中统计分数功能是否正确，这里只对“中”级进行测试演示。根据测试用例文档，其测试步骤如下所述。

（1）选择“游戏设置”|“游戏等级”|“中”命令，如图 11.18 所示。

（2）控制贪吃蛇吃到两个果实。查看游戏分数是否为“8”，如图 11.19 所示。



图 11.18　选择“中”命令

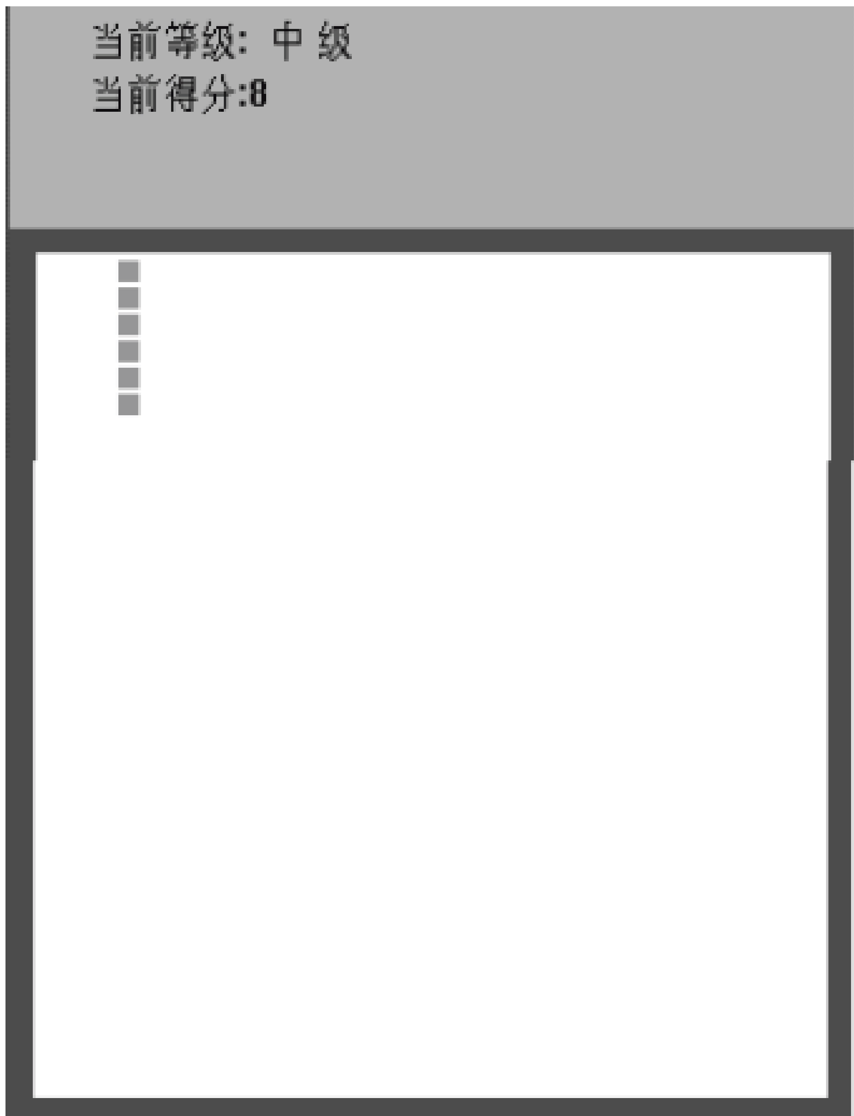


图 11.19　游戏分数统计显示

判断结果：贪吃蛇游戏分数统计功能正确。填写测试用例编号 1.7.3 结果为“通过”。

11.8　总　　结

本章通过贪吃蛇游戏实例项目开发的学习，希望各位读者能够更加深入地掌握好游戏项目开发中各种文档的格式及编写方法。同时知道自己如何开发一款贪吃蛇游戏。这个贪吃蛇游戏最核心的算法，是贪吃蛇的行走路径算法。其包含了数组应用、图像显示、分数计算等内容，请读者一定要仔细阅读和理解。

第 12 章　俄罗斯方块游戏项目开发

学习完第 11 章的贪吃蛇游戏，本章继续介绍俄罗斯方块游戏项目的开发。本章的内容结构和第 11 章大致相同。主要是为了帮助读者继续增加实际项目的开发经验。

本章主要涉及的内容如下：

- 俄罗斯方块项目的需求分析。
- 俄罗斯方块游戏的概要设计。
- 俄罗斯方块游戏操作界面设计。
- 俄罗斯方块游戏的详细设计及代码。
- 俄罗斯方块游戏的测试用例文档的编写及测试演示。

12.1　俄罗斯方块游戏项目的需求分析

所有的项目都是从需求分析开始做起的。需求获得和分析是项目开发的基础，只有获得明确的需求做出好的需求分析，才是保证项目成功的基础。

技巧：需求的获得不仅要由用户自己想到，还需要由分析者来引导用户。

12.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到俄罗斯方块游戏开发的资料如下所述。

（1）俄罗斯方块游戏的由来

顾名思义，俄罗斯方块自然是俄罗斯人发明的，这位伟人名叫阿列克谢・帕基特诺夫（Alexey Pazhitnov）。俄罗斯方块原名是俄语 Тетрис（英语是 Tetris），这个名字来源于希腊语 tetra，意思是“四”，而游戏的作者最喜欢网球（tennis），于是，他把两个词 tetra 和 tennis 合二为一，命名为 Tetris，这也就是俄罗斯方块名字的由来。

（2）俄罗斯方块游戏的操作方法

游戏通过“上”方向键来控制方块的变化，通过“左”、“右”和“下”方向键来移动从上落下的方块。

（3）俄罗斯方块游戏的基本规则

预先设置随机发生器不断地输出单个方块到游戏界面的顶部，以一定的规则进行移动、旋转、下落和摆放，并填充到游戏界面下面的方块中。每次摆放如果将下面方块的一行或多行完全填满，则组成这些行的所有小正方形将被消除，并且以此来换取一定的积分奖励。而未被消除的方块会一直累积，并对后来的方块摆放造成各种影响。如果未被消除

的方块堆放的高度超过游戏所规定的最大高度则游戏结束。

（4）英雄榜的显示及更新

当有玩家得到的分数超过当前记录分数线时，就把分数保存下来，在结束游戏时，要求玩家把名字保存下来。游戏初始时记录分数线为 0。例如，当第一个玩家得分为 10 分，结束游戏时，那么这个玩家的记录分将被保存下来并作为记录分数线。直到有玩家的得分超过 10 分，才能更新当前记录分数线并在退出游戏时保存玩家的分数及名字。

（5）游戏难度可以选择

在游戏开始前，可以选择游戏的难度，难度越高速度越快。难度共分为 10 级。而且根据游戏中分数不断的上升，游戏的难度会越来越大直到 10 级为止。

（6）可选择播放游戏背景音乐

在游戏开始后，可以选择播放背景音乐。

（7）游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

12.1.2　对语言描述进行需求分析

根据《俄罗斯方块用户需求描述文档》中的内容。现在需要对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

技巧：需求分析文档应做到对大部分的功能需求进行详细描述。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类俄罗斯方块游戏。特制定本说明书，用于描述某公司俄罗斯方块项目开发的功能性需求。

1.1　编写目的

使用技术性语言，对某公司的俄罗斯方块游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含某公司俄罗斯方块游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司俄罗斯方块游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

《俄罗斯方块用户需求描述文档》。

5．游戏具有的功能

5.1　能够显示主菜单和界面

游戏需要提供主菜单来让玩家进行游戏设置，同时能够显示当前分数、游戏等级等相

关信息到界面上。

5.2　能够实现方块的随机变化

随机生成不同样式的俄罗斯方块，方块的形状至少有 7 种以上变化。

5.3　能够控制方块的旋转、移动及落下

游戏以键盘进行操作，键盘上的“左”和“右”键用来控制落下砖块的左右移动；键盘上的“下”键用来控制砖块直接落到底；键盘上的“上”键用来控制砖块变形。

5.4　实现游戏规则

每当利用游戏中的方块填充完一行或者多行时，则可以消除当前这一行或者多行的方块。当方块堆放的高度超过游戏所规定的最大高度，即游戏界面最上面的边界时，则认定游戏结束。

5.5　游戏初始等级选择

通过主菜单，让玩家在游戏开始前，可以选择俄罗斯方块游戏的初始游戏等级。等级越高俄罗斯方块下移的速度就越快，消行得分也就越高。

5.6　游戏升级功能

在游戏中每消除 30 行方块，游戏等级增加一个等级，直到最大等级 10 级为止。而每一个等级的时间间隔如表 12.1 所示。

表 12.1　游戏等级与时间对照

游 戏 等 级	时 间 间 隔	游 戏 等 级	时 间 间 隔
1	300ms	6	200ms
2	280ms	7	180ms
3	260ms	8	160ms
4	240ms	9	140ms
5	220ms	10	120ms

5.7　分数统计功能

在游戏中每消除一行或者多行方块，游戏分数就可以进行相应的增加。其分数计算公式如下：

```
消除一行得分 = 10 分+等级*10 分
消除二行得分 = 30 分+等级*10 分
消除三行得分 = 50 分+等级*10 分
消除四行得分 = 100 分+等级*10 分
游戏总数 = 累计消除行数得分
```

5.8　英雄榜的更新

当有玩家得到的分数超过当前记录分数线时，就把分数保存下来，在结束游戏时，要求玩家把名字保存下来。游戏初始时记录分数线为 0。

例如，当第一个玩家得分为 10 分，结束游戏时，那么这个玩家的记录分将被保存下来并作为记录分数线。直到有玩家的得分超过 10 分，才能更新当前记录分数线，并在退出游戏时保存玩家的分数及名字。

5.9　游戏支持背景音乐播放功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.10　游戏提供帮助说明

在游戏菜单中，提供一个使用说明项，以方便对本游戏不了解的玩家对游戏进行操作和使用。

12.2　俄罗斯方块游戏概要设计

笔者编写的俄罗斯方块游戏的概要设计文档内容如下所述。

1．引言

1.1　编写目的

为了让每个开发人员明白俄罗斯方块游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．术语

3．参考文献

《俄罗斯方块游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高键盘操作易用性。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。
- Microsoft® Visual C++ 6.0。

4.3　需求概述

参见《俄罗斯方块游戏需求分析说明书》。

4.4　条件与限制

无。

5．总体设计

5.1　俄罗斯方块游戏的功能架构（如图 12.1 所示）

5.2　各功能处理流程

内容参见《俄罗斯方块游戏各功能详细设计文档》。

6．接口设计

内容参见《俄罗斯方块游戏操作界面设计文档》。

7．类结构设计

游戏由 7 个类组成，如图 12.2 所示。

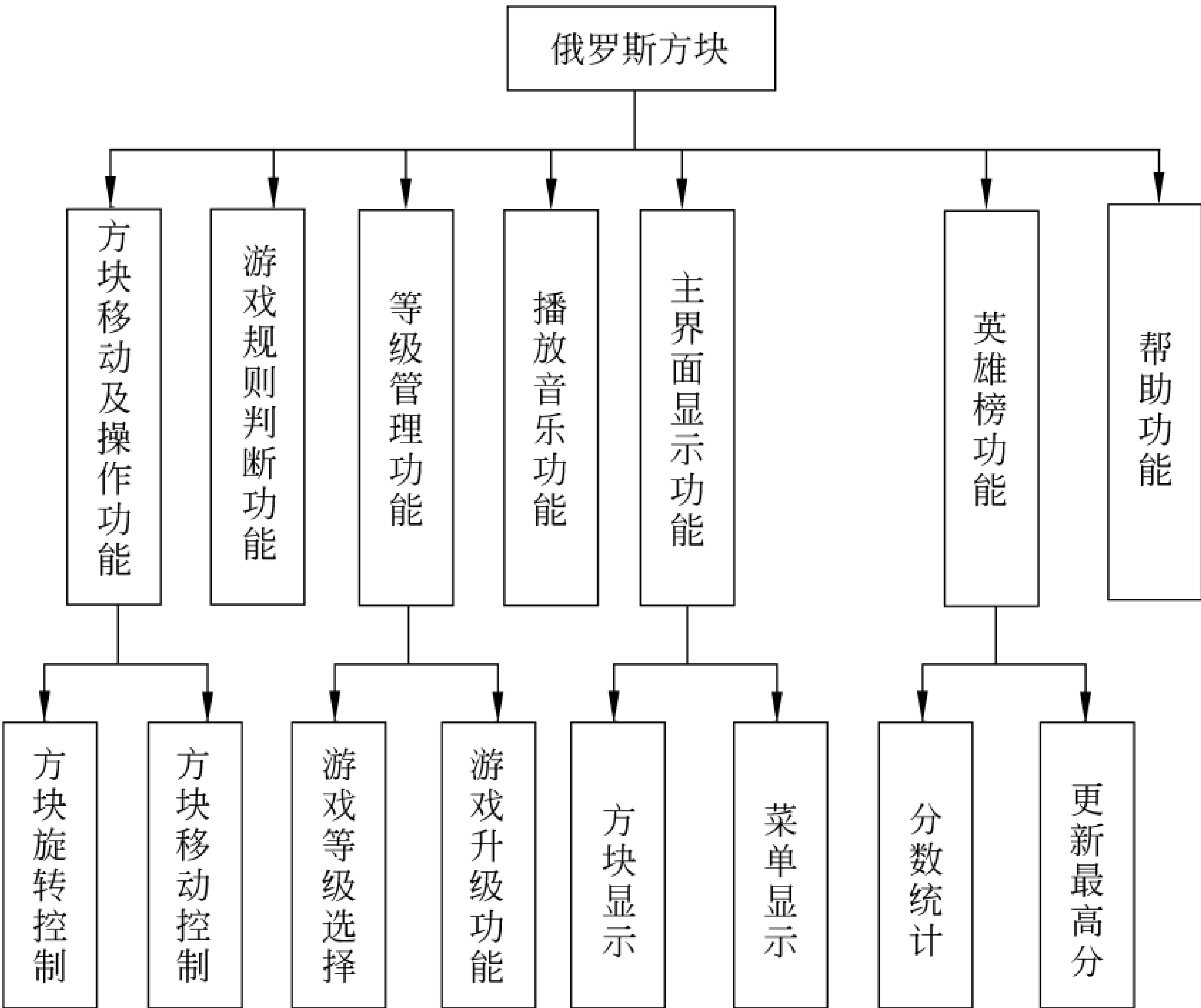


图 12.1　俄罗斯方块功能架构

- 主界面对话框类：主要负责主界面及菜单的显示，同时负责调用其他接口函数。
- 游戏规则类：主要负责游戏规则的实现。
- 方块游戏类：主要负责俄罗斯方块的移动、下落等操作，同时还要负责分数、等级的显示。
- 英雄榜对话框类：主要负责游戏分数的统计及高分记录的更新。
- 音乐播放类：主要负责游戏中背景音乐的播放。
- 帮助对话框类：主要负责帮助提示的显示及其他辅助信息。
- 等级设置对话框类：主要负责当前游戏等级的设置。

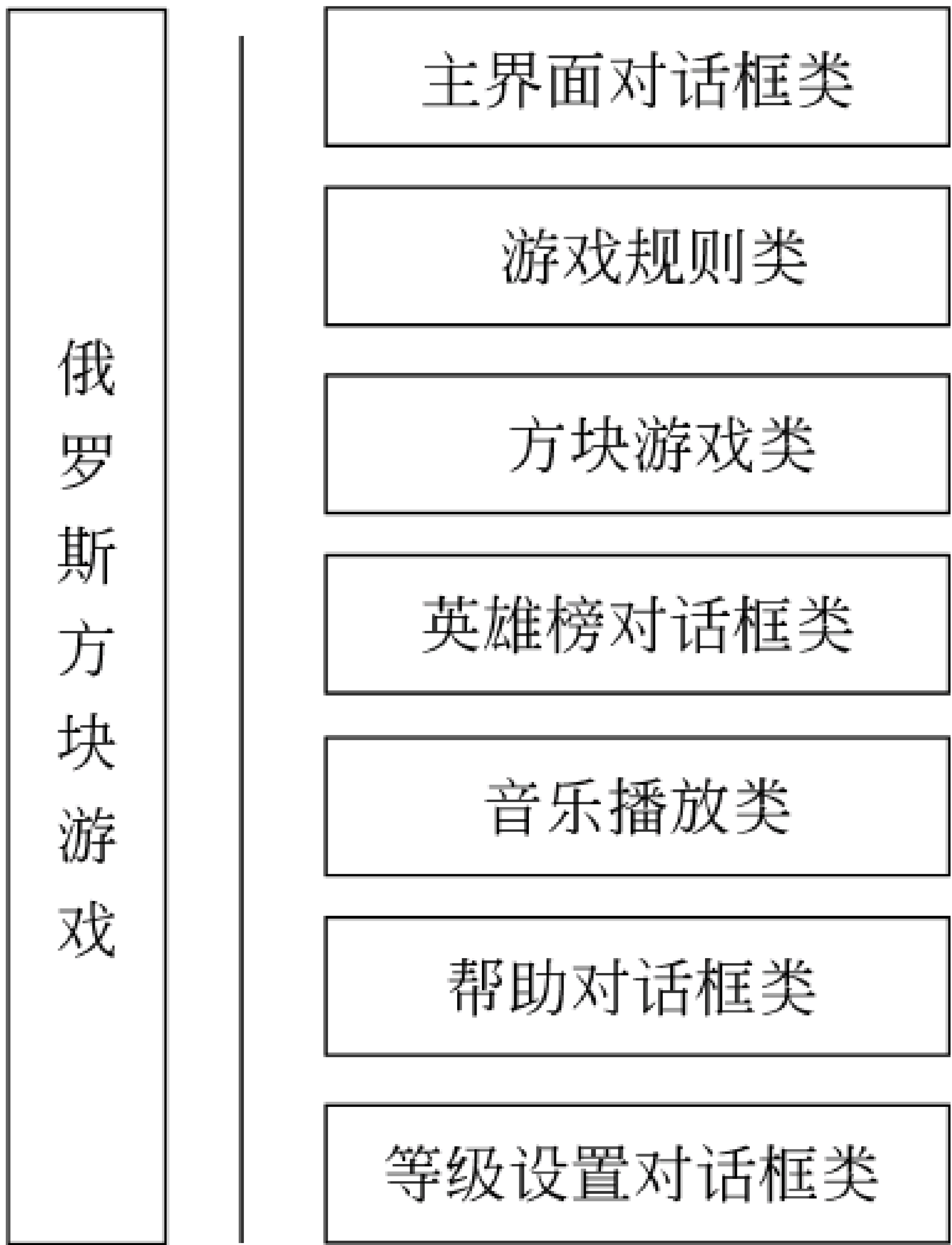


图 12.2　游戏主要类结构

8．出错处理设计

8.1　出错输出信息

当游戏中出现错误，采用弹出对话框的方式来提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误时，采用中止当前游戏并重新开始新游戏的方法来处理游戏中的错误。

9．维护设计

由于整个俄罗斯方块游戏项目在开发完成后，基本不会有太多的变动，所以主要的维护任务是把用户使用中出现的问题解决。

12.3　俄罗斯方块游戏操作界面及测试用例设计

本章将继续介绍《游戏操作界面设计文档》和《游戏测试用例文档》的编写。

12.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的俄罗斯方块游戏的操作界面设计文档内容如下所示。

技巧：详细的操作界面设计文档可以让用户提前知道操作界面是否满足其要求。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确俄罗斯方块游戏的操作界面是如何设计的，特制定本文档来用于描述本公司俄罗斯方块游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含本公司俄罗斯方块游戏的操作界面设计。

3．使用对象

本说明书的使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《俄罗斯方块游戏需求分析说明书》和《俄罗斯方块游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面的设计

俄罗斯方块的游戏主界面设计，如图 12.3 所示。

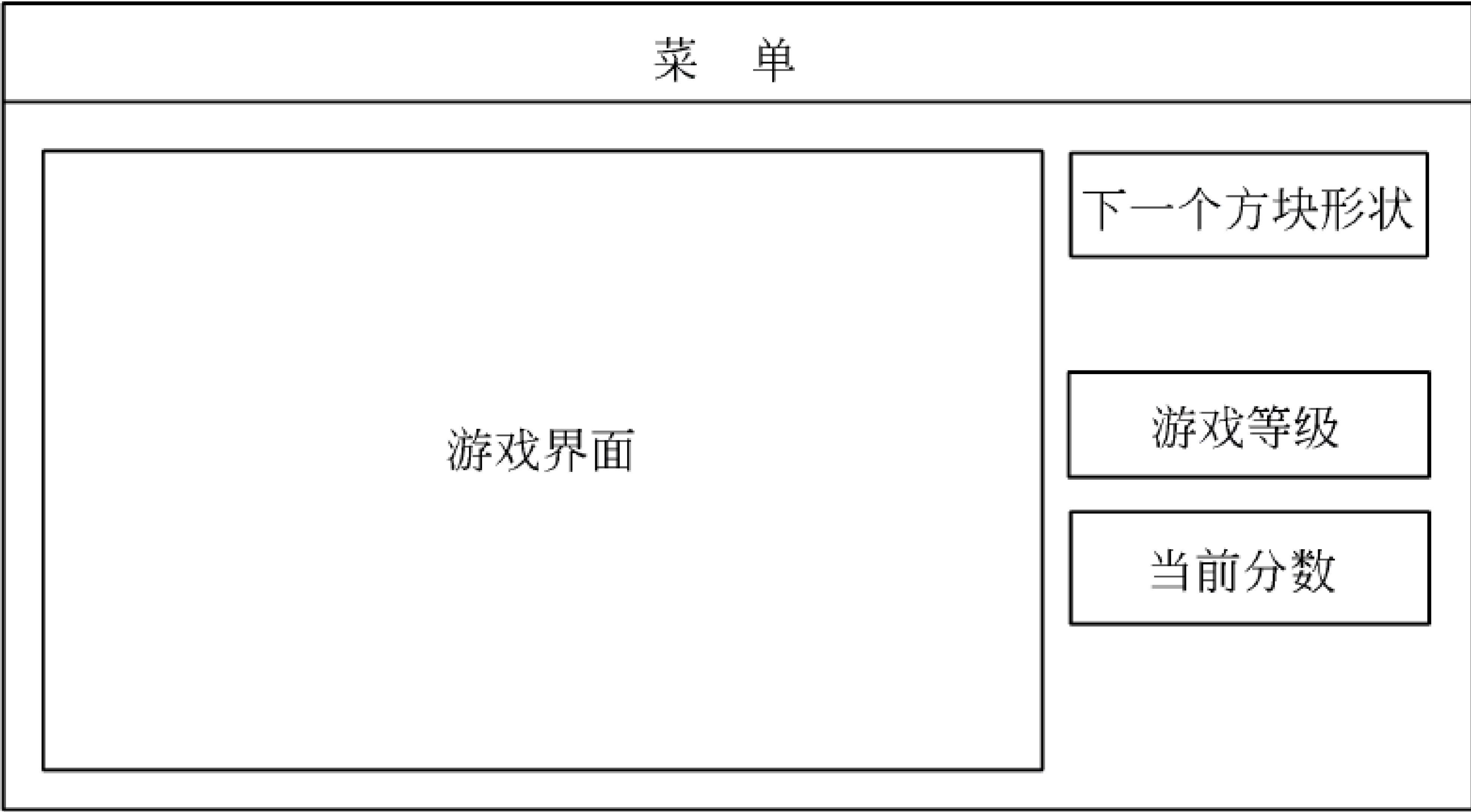


图 12.3　设计的游戏主界面

5.2　游戏菜单结构的设计

俄罗斯方块的游戏菜单设计如图 12.4 所示。

5.3　游戏等级设置对话框的设计

俄罗斯方块游戏的等级设置对话框的设计如图 12.5 所示。

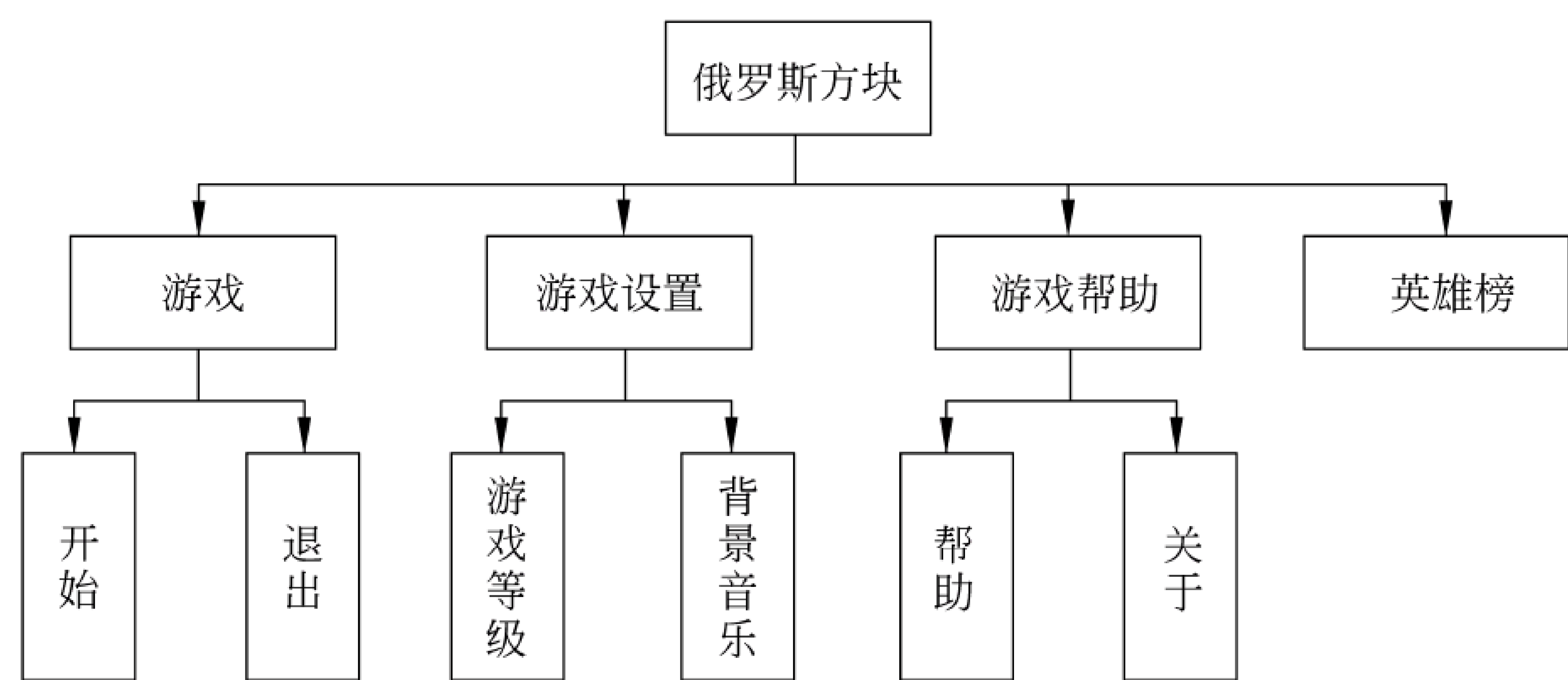


图 12.4　设计的游戏菜单结构

5.4　英雄榜对话框的设计

俄罗斯方块英雄榜对话框的设计如图 12.6 所示。

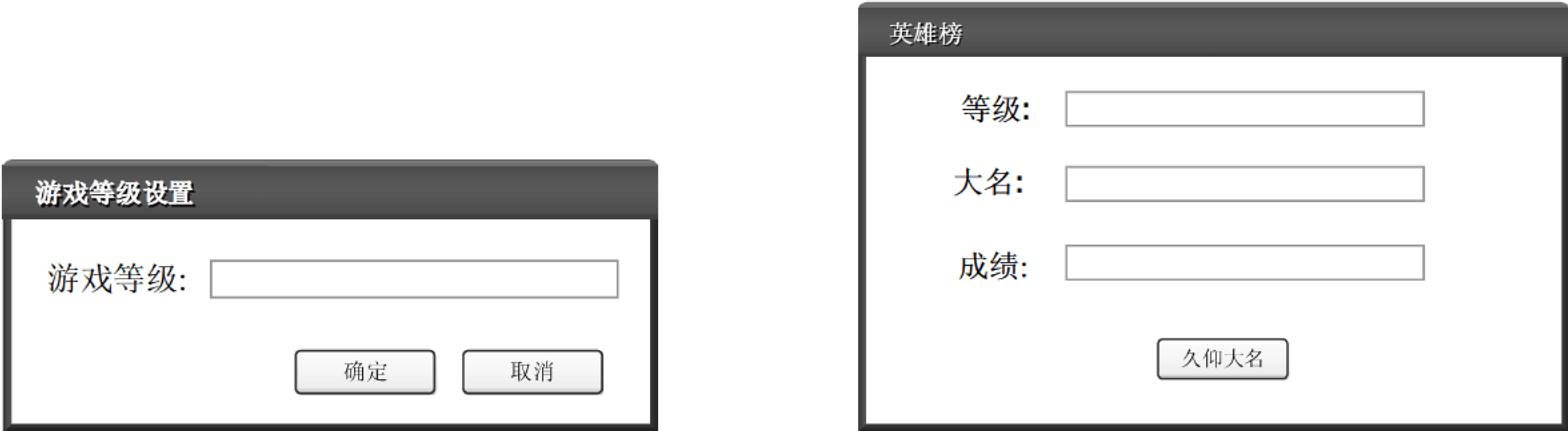


图 12.5　设计的游戏等级设置对话框　　　图 12.6　设计的英雄榜对话框

12.3.2　测试用例文档

测试用例文档主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的俄罗斯方块游戏的测试用例文档内容如下所述。

技巧： 测试用例文档必须清楚地描述测试目的、测试的过程及预期目的，这样才能让测试人员有的放矢。

1．引言

本文档主要用于某公司在开展俄罗斯方块游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了俄罗斯方块游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B 和 C 三类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过。方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的俄罗斯方块游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行俄罗斯方块游戏项目的测试和验收。

3．使用对象

本测试用例文档的使用对象主要是与某公司俄罗斯方块游戏开发相关的需求分析、测试和维护等部门（单位）的人员。

4．参考文献

《俄罗斯方块游戏的需求分析说明书》；

《俄罗斯方块游戏的概要设计文档》；

《俄罗斯方块游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试项目

测试项目主要针对俄罗斯方块中的各种功能进行整合性测试，共包含如下几个项目。

（1）主菜单和界面显示功能的测试，其主要内容如表 12.2 所示。

表 12.2　主菜单和界面显示功能的测试

测试编号：1.7.1	类别：A
项　　目：俄罗斯方块测试	
分 项 目：主菜单和界面显示功能的测试	
测试目的：测试俄罗斯方块游戏中的菜单和界面是否正确显示	
测试配置：	
预置条件： 俄罗斯方块游戏源程序已经编译完成，并可以运行； 键盘和鼠标已准备好	
测试步骤： 运行俄罗斯方块程序，查看菜单和界面	
预期结果： 游戏主界面及菜单与操作设计文档中的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏主界面和菜单是否正确显示（是/否）	
测试结果： 通过/不通过	

（2）方块的随机变化功能的测试，其主要内容如表 12.3 所示。

表 12.3　方块的随机变化功能的测试

测试编号：1.7.2	类别：A
项　　目：俄罗斯方块测试	

续表

分 项 目：方块的随机变化功能的测试
测试目的：测试游戏中方块出现的样式是否随机变化
测试配置：
预置条件： 俄罗斯方块游戏已经开始
测试步骤： 连续落下 7 个方块后，查看其中至少 3 个方块的样式
预期结果： 至少 3 个方块的样式不相同
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏中方块的出现的样式是否随机变化（是/否）
测试结果： 通过/不通过

（3）方块移动功能的测试，其主要内容如表 12.4 所示。

表 12.4　方块移动功能的测试

测试编号：1.7.3	类别：A
项　　目：俄罗斯方块测试	
分 项 目：方块移动功能的测试	
测试目的：测试游戏中的方块是否能够按照正确的方向移动并旋转	
测试配置：	
预置条件： 俄罗斯方块游戏源程序已经编译完成，并可以运行	
测试步骤： 分别单击“左”、“右”按键，查看方块； 单击“上”按键，查看方块； 单击“下”按键，查看方块	
预期结果： 单击“左”按键时，方块向左移动； 单击“右”按键时，方块向右移动； 单击“上”按键时，方块进行旋转变化； 单击“下”按键时，方块直接落下到底	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 方块是否能够按照正确的方向移动并旋转（是/否）	
测试结果： 通过/不通过	

（4）游戏等级选择功能的测试，其主要内容如表 12.5 所示。

表 12.5　游戏等级选择功能的测试

测试编号：1.7.4	类别：A
项　　目：俄罗斯方块测试	
分 项 目：游戏等级选择功能的测试	
测试目的：测试游戏中的等级是否可以设置	
测试配置：	
预置条件： 键盘准备好； 游戏已经开始	
测试步骤： 选中“游戏设置”|“游戏等级”； 在弹出的对话框中，指定当前游戏等级； 查看主界面中，游戏提示的当前等级	
预期结果： 主界面中，出现的游戏等级提示与选择的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的等级是否可以设置（是/否）	
测试结果： 通过/不通过	

（5）游戏规则功能的测试，其主要内容如表 12.6 所示。

表 12.6　游戏规则功能的测试

测试编号：1.7.5	类别：A
项　　目：俄罗斯方块测试	
分 项 目：游戏规则功能的测试	
测试目的：测试俄罗斯方块游戏能否对结束和消行规则进行判断	
测试配置：	
预置条件： 游戏已经开始； 游戏等级已经设定	
测试步骤： 让方块填充完一行或者多行，查看当前游戏反应； 让方块一直积累到游戏界面的最上方。查看当前游戏的反应	
预期结果： 当填充完一行或者多行时，会自动消除掉，并增加分数； 当方块积累到最上方时，提示游戏结束	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	

续表

测试记录： 俄罗斯方块游戏是否能对结束和消行规则进行判断（是/否）
测试结果： 通过/不通过

（6）背景音乐播放功能的测试，其主要内容如表 12.7 所示。

表 12.7　背景音乐播放功能的测试

测试编号：1.7.6	类别：A
项　　目：俄罗斯方块测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试俄罗斯方块游戏能否支持播放背景音乐	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否支持播放背景音乐（是/否）	
测试结果： 通过/不通过	

（7）帮助功能的测试，其主要内容如表 12.8 所示。

表 12.8　帮助功能的测试

测试编号：1.7.7	类别：A
项　　目：俄罗斯方块测试	
分 项 目：帮助功能的测试	
测试目的：测试俄罗斯方块游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选中“游戏帮助”|“帮助”菜单栏	
预期结果： 出现游戏帮助提示，说明游戏操作方法	

续表

判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 俄罗斯方块游戏是否有帮助提示功能（是/否）
测试结果： 通过/不通过

（8）英雄榜功能的测试，主要内容如表 12.9 所示。

表 12.9　英雄榜功能的测试

测试编号：1.7.8	类别：A
项　　目：俄罗斯方块测试	
分　项　目：英雄榜功能的测试	
测试目的：测试俄罗斯方块游戏的英雄榜记录是否正确	
测试配置：	
预置条件： 玩家在游戏中已经超过上次最高记录分数	
测试步骤： 等待游戏结束； 在弹出的对话框中输入玩家的大名； 选中“英雄榜”菜单栏	
预期结果： 在弹出的对话框中，查看等级、大名及分数是否被记录	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 俄罗斯方块游戏的英雄榜记录是否正确（是/否）	
测试结果： 通过/不通过	

（9）游戏升级功能的测试，主要内容如表 12.10 所示。

表 12.10　游戏升级功能的测试

测试编号：1.7.9	类别：A
项　　目：俄罗斯方块测试	
分　项　目：游戏升级功能的测试	
测试目的：测试游戏中的自动升级功能是否正确	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 连续消除 30 行方块，查看当前游戏等级	

续表

预期结果： 当前游戏等级比开始玩时的等级高一级，但如果到达 10 级时，就不再升级，而是游戏通关的提示对话框出现
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 俄罗斯方块游戏中的自动升级功能是否正确（是/否）
测试结果： 通过/不通过

（10）游戏计分功能的测试，主要内容如表 12.11 所示。

表 12.11　游戏计分功能的测试

测试编号：1.7.10	类别：A
项　　目：俄罗斯方块测试	
分 项 目：游戏计分功能的测试	
测试目的：测试游戏中的计分功能是否正确	
测试配置：	
预置条件： 游戏已经运行； 设置当前游戏等级为“1”	
测试步骤： 消除“1”行方块，查看当前游戏得分； 消除“2”行方块，查看当前游戏得分； 消除“3”行方块，查看当前游戏得分； 消除“4”行方块，查看当前游戏得分	
预期结果： 当前游戏分数显示为“20”； 当前游戏分数显示为“40”； 当前游戏分数显示为“60”； 当前游戏分数显示为“110”	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 俄罗斯方块游戏中计分功能是否正确（是/否）	
测试结果： 通过/不通过	

12.4　俄罗斯方块游戏的详细设计

本节主要对俄罗斯方块游戏中的功能进行详细讲解。为了突出重点，在这里不描述详细设计文档的格式。

🔔技巧：详细设计文档必须按照设计者的思路进行描述，才能在编码阶段少走弯路。

12.4.1　游戏各功能的设计描述

在俄罗斯方块游戏中，大致可以分为 7 个功能模块，如下所示。

1．游戏移动模块的算法设计

游戏移动模块的算法主要分为如下几个步骤：

（1）接收玩家键盘输入消息，判断所按方向按键，并保存在当前方向变量中。

（2）当方向为“左”或者“右”时，判断是否碰壁，如果是，则不进行左右移动。如果不是，则把保存方块位置数组中的坐标数据进行相应更新。

（3）向下移动，判断是否落到最下一行。如果不是，则把保存方块位置数组中的坐标数据进行相应更新。如果是，则继续判断是否填充这一行或者多行为完整一行或者多行，如果是，则把这一行或者多行进行消除；如果不是，则更新整个显示方块数组中的数据。

2．方块生成模块的算法设计

方块生成模块的算法主要分为如下几个步骤：

（1）使用对生成的随机数取余，得到当前生成方块类型值。

（2）用方块类型值，去生成相应的方块。

（3）当前生成的方块数据更新到方块数组中。

3．游戏规则模块的算法设计

当方块向下时，判断当前方块数组中的元素是否有超过上边界的，如果有，则游戏结束；如果没有，则继续游戏。

4．游戏等级设置模块的算法设计

游戏等级设置模块的算法主要分为如下几个步骤：

（1）在玩家选中“游戏设置”|“游戏等级”菜单栏时，生成游戏等级设置对话框，并读取配置文件。

（2）在玩家修改完成后，直接把相关设置保存到配置文件中。在开始游戏时，直接读取配置文件 setup.ini，得到并显示当前游戏等级。

5．英雄榜模块的算法设计

英雄榜模块的算法主要分为如下几个步骤：

（1）读取配置文件（setup.ini），得到并显示当前最高分记录、大名及等级。

（2）在用户结束游戏时，比较当前用户得分和最高分。如果高于最高分，就弹出“英雄榜”对话框，要求用户输入大名，并连同用户的等级和分数保存到配置文件中。

6．音乐播放模块的算法设计

音乐播放模块比较简单，只需要在用户选择音乐播放时，把音乐资源载入程序并播放。

7．帮助类模块的算法设计

帮助类模块的算法也比较简单，只要把相应的对话框资源显示出来即可。

12.4.2　游戏的各功能流程图

在这里笔者只给出了移动、绘制方块功能的详细流程图，如图 12.7 所示。

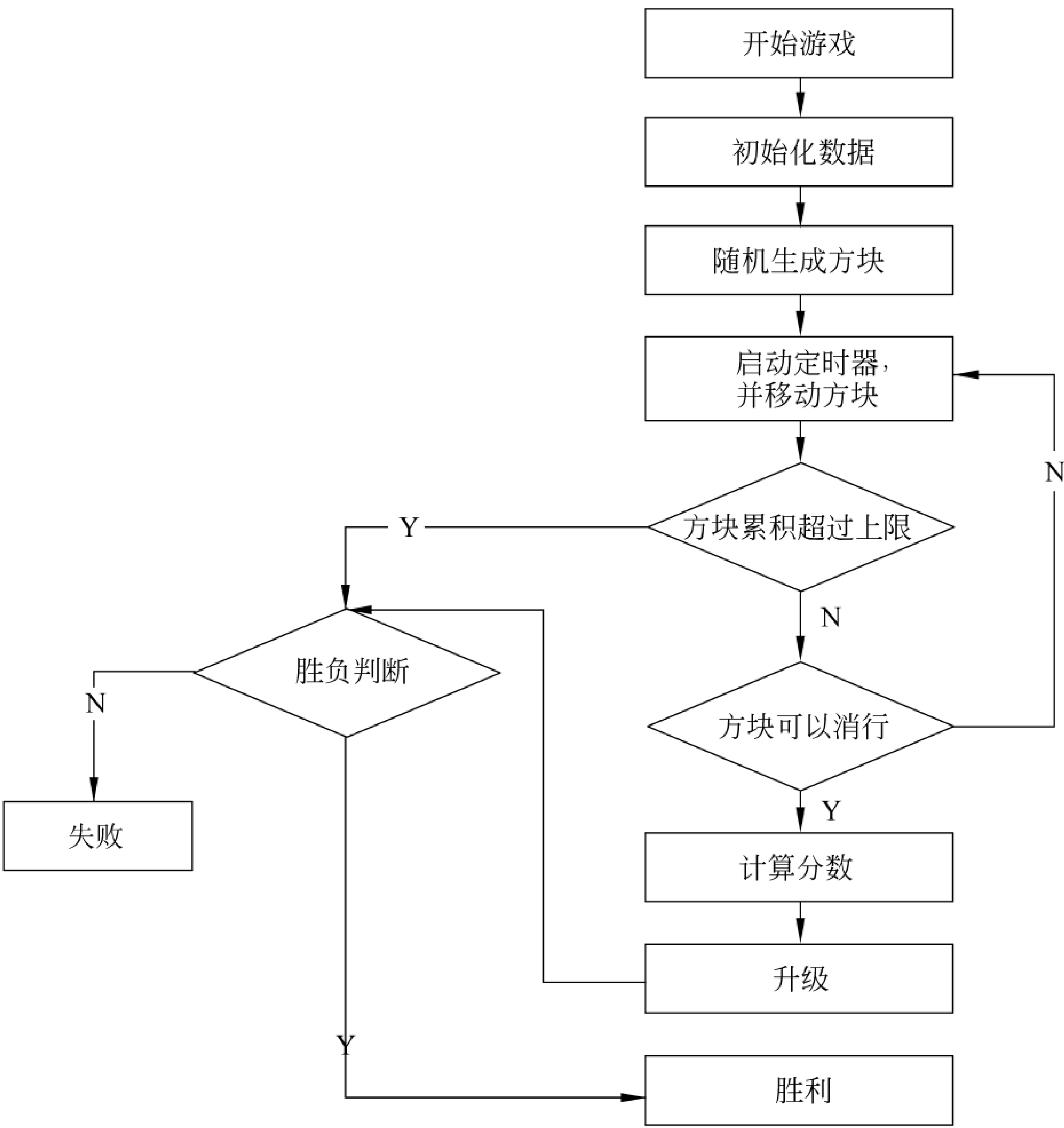


图 12.7　移动和绘制方块功能流程

12.5　俄罗斯方块游戏的界面实现

俄罗斯方块游戏的 Visual C++工程采用 MFC 框架模式。本节将主要讲解俄罗斯方块游戏的各个功能模块的代码实现。

12.5.1　游戏菜单的实现

在俄罗斯方块游戏中，通过如下几个步骤即可实现游戏的菜单。

（1）在俄罗斯方块游戏工程的资源中添加一个菜单资源，其属性如表 12.12 所示。

表 12.12　主菜单属性

ID	类　　别	说　　明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
ID_APP_EXIT	菜单栏	退出游戏
IDR_LEVEL_SETUP	弹出菜单	游戏等级
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于
IDR_HERO_LIST	菜单栏	英雄榜

（2）给每个菜单栏添加响应函数到 CTetrisView 类中。

（3）菜单响应函数应尽量调用类中的其他功能函数，减少直接处理的过程。这样程序代码阅读起来结构简单，功能明确。菜单响应函数的实现，如代码 12.1 所示。

代码 12.1　菜单响应函数的实现

```
01  // TetrisView.cpp CTetrisView 类的源文件
02
03
04  #include "stdafx.h"
05  #include "Tetris.h"
06
07  #include "TetrisDoc.h"
08  #include "TetrisView.h"
09
10  #include "HelpDlg.h"                    //插入帮助对话框类声明头文件
11  #include "HeroDlg.h"                    //插入英雄榜对话框类声明头文件
12  #include "LevelSetupDlg.h"              //插入等级设置对话框类声明头文件
13  #include "russia.h"                     //插入主游戏类头文件
14
15  ////////////////////////////////////////
16  // CtetrisView 视图类实现
17
18  IMPLEMENT_DYNCREATE(CTetrisView, CView)
19                                          //消息与函数映射
20  BEGIN_MESSAGE_MAP(CTetrisView, CView)
21      ON_COMMAND(IDR_ABOUT, OnAbout)      //资源与函数映射响应关系
22      ON_COMMAND(IDR_HERO_LIST, OnHeroList)
23      ON_COMMAND(IDR_LEVEL_SETUP, OnLevelSetup)
24      ON_COMMAND(IDR_PLAY_MUSIC, OnPlayMusic)
25      ON_COMMAND(IDR_START_GAME, OnStartGame)
26      ON_COMMAND(IDR_HELP, OnHelp)
27      ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
28      ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
29      ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
30  END_MESSAGE_MAP()
31
32  ///////////////////////////////////
33  // CTetrisView 的构造与析构函数
34
35  CTetrisView::CTetrisView()              //构造函数
36  {
```

```
37  }
38  CTetrisView::~CTetrisView()               //析构函数
39  {
40  }
41                                            //建立窗口前被调用的函数
42  BOOL CTetrisView::PreCreateWindow(CREATESTRUCT& cs)
43  {
44      return CView::PreCreateWindow(cs);
45  }
46
47  /////////////////////////////////////////
48  // CTetrisView 类的绘图函数
49
50  void CTetrisView::OnDraw(CDC* pDC)
51  {
52      CTetrisDoc* pDoc = GetDocument();
53      ASSERT_VALID(pDoc);
54  }
55
56  BOOL CTetrisView::OnPreparePrinting(CPrintInfo* pInfo)
57  {
58      return DoPreparePrinting(pInfo);
59  }
60                                            //开始打印函数
61  void CTetrisView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
62  {
63  }
64                                            //停止打印函数
65  void CTetrisView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
66  {
67  }
68
69  ////////////////////////////////////////////
70  // CTetrisView 类的诊断函数
71
72  #ifdef _DEBUG
73  void CTetrisView::AssertValid() const
74  {
75      CView::AssertValid();
76  }
77
78  void CTetrisView::Dump(CDumpContext& dc) const
79  {
80      CView::Dump(dc);
81  }
82
83  CTetrisDoc* CTetrisView::GetDocument() //内联函数，获得文档指针
84  {
85      ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTetrisDoc)));
86      return (CTetrisDoc*)m_pDocument;
87  }
88  #endif //_DEBUG
89
90  /////////////////////////////////////////
91  // CTetrisView 消息响应函数实现
92  void CTetrisView::OnAbout()
93  {
94      CAboutDlg aboutDlg;                   //生成关于对话框
95      aboutDlg.DoModal();                   //弹出关于对话框
```

```
96  }
97
98  void CTetrisView::OnHeroList()
99  {
100     CHeroDlg dlg;                           //生成英雄榜对话框
101     dlg.DoModal();                          //弹出英雄榜对话框
102 }
103
104 void CTetrisView::OnLevelSetup()
105 {
106     CLevelSetupDlg  dlg;                    //生成等级设置对话框
107     dlg.DoModal();                          //弹出等级设置对话框
108 }
109
110 void CTetrisView::OnPlayMusic()
111 {
112     CWnd*   pMain   =   AfxGetMainWnd();
113     CMenu*   pMenu   =   pMain->GetMenu();
114     //判断播放音乐菜单当前状态
115     BOOL bCheck = (BOOL)pMenu->GetMenuState(IDR_PLAY_MUSIC, MF_CHECKED);
116
117     if(m_bStart)
118     {                                       //游戏如果开始才允许播放音乐
119         if(bCheck)
120         {
121             pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
122                             MF_BYCOMMAND | MF_UNCHECKED);
123         }
124         else
125         {
126             pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
127                            MF_BYCOMMAND | MF_CHECKED);
128         }
129
130         PlayBackMusic(!bCheck);             //调用播放背景音乐功能函数
131     }
132
133 }
134
135 void CTetrisView::OnStartGame()
136 {
137     russia.GameStart();                     //调用 russia 对象的游戏开始函数
138 }
139
140 void CTetrisView::OnHelp()
141 {
142     CHelpDlg dlg;                       //生成帮助对话框
143     dlg.DoModal();                      //弹出对话框
144 }
```

代码解析：代码第 137 行，是通过调用对象 russia 的函数 GameStart()来实现游戏开始功能。

12.5.2　游戏帮助对话框的实现

俄罗斯方块游戏中的帮助是使用一个对话框来实现的，其实现步骤如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 12.8 所示。

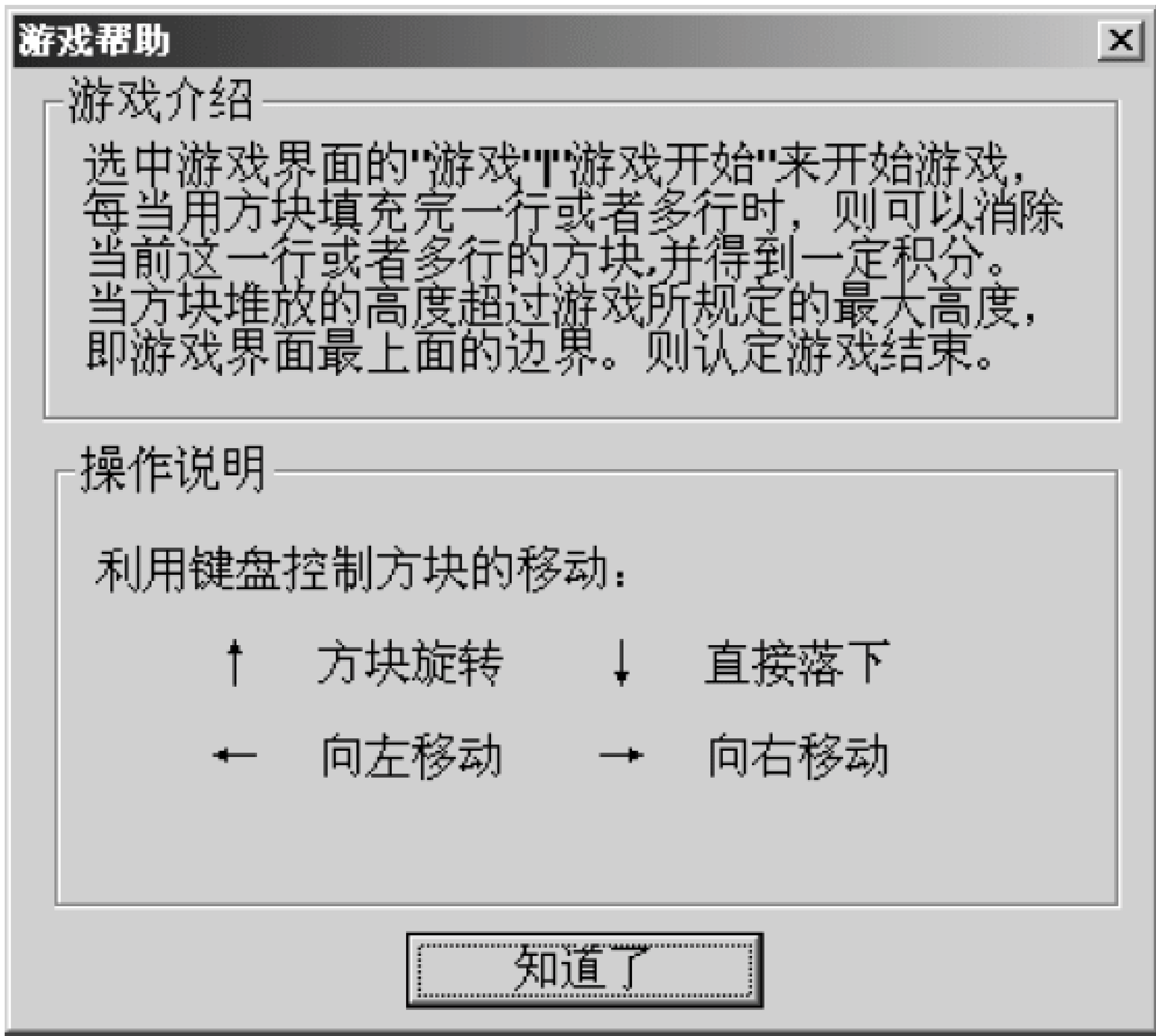


图 12.8　帮助对话框

（2）编写一个 CHelpDlg 对话框类，其中主要是加载 IDD_HELP 对话框资源，通过资源中的文字说明对游戏操作方法进行描述。其代码如代码 12.2 所示。

代码 12.2　CHelpDlg 对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
04  // HelpDlg.h CHelpDlg类声明头文件
05
06
07  ///////////////////////////////////////
08  // CHelpDlg 对话框类
09
10  class CHelpDlg : public CDialog          //公共继承于 CDialog 类
11  {
12  public:
13      CHelpDlg(CWnd* pParent = NULL);     //构造函数
14
15  //对话框资源
16      enum { IDD = IDD_HELP };             //加载资源
17
18  //重载函数
19      protected:
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                 //单击“确定”按钮响应函数声明
25      DECLARE_MESSAGE_MAP()
26  };
27
28  #endif
```

（3）CHelpDlg 对话框类的实现比较简单，就是一个基本对话框类的结构。只对其中的“知道了”按钮响应函数进行实现，用于退出当前对话框。其代码如代码 12.3 所示。

代码 12.3　CHelpDlg 对话框类的实现

```
01  // HelpDlg.cpp CHelpDlg 类的实现源文件
02
03
04  #include "stdafx.h"                          //插入头文件
05  #include "Tetris.h"
06  #include "HelpDlg.h"                         //插入类声明头文件
07
08  ////////////////////////////////////////////
09  // CHelpDlg 对话框类实现
10
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/)//构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);           //调用基类加载函数
19  }
20
21  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)
22  END_MESSAGE_MAP()
23
24  ////////////////////////////////////////////
25  // CHelpDlg 消息响应函数
26
27  void CHelpDlg::OnOK()                        //单击“知道了”按钮响应函数
28  {
29      CDialog::OnOK();
30  }
```

12.5.3　游戏英雄榜对话框的实现

俄罗斯方块游戏英雄榜的实现，分为如下几个步骤。

（1）创建一个对话框资源，并添加相应的控件，如图 12.9 所示。

（2）配置（setup.ini）文件格式如下：

```
[HERO]
name=XXX
score=0
level=0
```



图 12.9　英雄榜对话框资源

（3）添加 CHeroDlg 类，其声明中包含了“设置可写记录标志”接口函数，用于外部函数调用时，设置是否对配置文件进行写操作。其代码如代码 12.4 所示。

代码 12.4　CHeroDlg 类的声明

```
01  #if !defined(AFX_HERODLG_H__)
02  #define AFX_HERODLG_H__
03
04  // HeroDlg.h 头文件
05
06  ///////////////////////////////////////////
07  // CHeroDlg 对话框类声明
08
09  class CHeroDlg : public CDialog
10  {
11  public:
12      void SetWriteFlg(BOOL bflg);                //接口函数，设置可记录标志变量
13      CHeroDlg(CWnd* pParent = NULL);             //构造函数
14
15      enum { IDD = IDD_HERO_LIST };               //对话框资源
16      int     m_level;                            //保存等级变量
17      CString m_name;                             //保存姓名变量
18      int     m_score;                            //保存分数变量
19
20      public:
21      virtual int DoModal();                      //弹出对话框函数声明
22      protected:
23      virtual void DoDataExchange(CDataExchange* pDX);
24
25  protected:
26
27      virtual void OnOK();                        //单击“久仰”按钮响应函数声明
28      DECLARE_MESSAGE_MAP()
29  private:
30      BOOL m_bWriteflg;                           //记录标志变量
31  };
32
33  #endif
```

（4）CHeroDlg 类的实现中通过调用系统 API 函数，对配置文件进行读写操作。而“设置读写标志”接口函数，是对类的一个成员变量 m_bWrite 进行赋值操作，达到写入或者读取的区分。其代码如代码 12.5 所示。

代码 12.5　CHeroDlg 类的实现

```
01  // HeroDlg.cpp 源文件
02  #include "stdafx.h"                             //插入头文件
03  #include "Tetris.h"
04  #include "HeroDlg.h"                            //插入类声明头文件
05
06  ///////////////////////////////////////////
07  // CHeroDlg 对话框
08
09  CHeroDlg::CHeroDlg(CWnd* pParent /*=NULL*/)//构造函数
10      : CDialog(CHeroDlg::IDD, pParent)
11  {
12      m_bWriteflg = FALSE;                        //初始化写标志变量为假
```

```
13  }
14
15  void CHeroDlg::DoDataExchange(CDataExchange* pDX)
16  {
17      CDialog::DoDataExchange(pDX);            //变量与资源映射
18      //{{AFX_DATA_MAP(CHeroDlg)
19      DDX_Text(pDX, IDC_LEVEL_EDIT, m_level);
20      DDX_Text(pDX, IDC_NAME_EDIT, m_name);
21      DDX_Text(pDX, IDC_SCORE_EDIT, m_score);
22      DDV_MinMaxInt(pDX, m_score, 0, 10000);
23      //}}AFX_DATA_MAP
24  }
25
26
27  BEGIN_MESSAGE_MAP(CHeroDlg, CDialog)
28      ON_BN_CLICKED(IDOK_BTN, OnBtn)           //按钮与函数映射
29  END_MESSAGE_MAP()
30
31  //////////////////////////////////////////////////////////////////////
32  // CHeroDlg 消息句柄
33
34  void CHeroDlg::SetWriteFlg(BOOL bflg)        //设置写入标志
35  {
36      m_bWriteflg = bflg;                      //设置读写标志
37  }
38
39  int CHeroDlg::DoModal()                      //弹出对话框
40  {
41      char pszTmp[128] = {0};
42
43      //读取配置文件
44      GetPrivateProfileString("HERO", "name", "0",
45          pszTmp, 127, ".\\hero.ini");         //读姓名
46      m_name = CString(pszTmp);
47
48      if(!m_bWriteflg)
49      {
50          GetPrivateProfileString("HERO", "score", "0",
51              pszTmp, 127, ".\\hero.ini");     //读分数
52          m_score = atoi(pszTmp);
53          GetPrivateProfileString("HERO", "level", "0",
54              pszTmp, 127, ".\\hero.ini");     //读等级
55          m_level = atoi(pszTmp);
56      }
57
58      return CDialog::DoModal();
59  }
60
61  void CHeroDlg::OnBtn()                       //按钮响应
62  {
63      UpdateData(TRUE);
64      if(m_bWriteflg)
65      {
66          CString tmp;
67          tmp.Format("%d", m_score);
68        WritePrivateProfileString("HERO", "name", m_name, ".\\hero.ini");
                                                 //写入姓名
69        WritePrivateProfileString("HERO", "score", tmp, ".\\hero.ini");
                                                 //写入分数
```

```
70          tmp.Format("%d", m_level);
71         WritePrivateProfileString("HERO", "level", tmp, ".\\hero.ini");
                                                        //写入等级
72      }
73      m_bWriteflg = FALSE;
74
75      CDialog::OnOK();
76  }
77
78  BOOL CHeroDlg::OnInitDialog()                       //初始化对话框
79  {
80      CDialog::OnInitDialog();
81
82      if(m_bWriteflg)
83      {                                               //当为写入时，改变按钮名称
84          SetDlgItemText(IDOK_BTN, "记录");
85      }
86
87      return TRUE;
88  }
```

代码解析：代码第 50 行是通过调用 API 函数 GetPrivateProfileString()实现对俄罗斯方块游戏配置文件内容的读取。代码第 68 行同样是调用 API 函数 WritePrivateProfileString()实现对游戏配置文件的内容的写入。

12.5.4　游戏播放背景音乐的实现

播放游戏背景音乐，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。但要注意，这里使用的这个 API 函数是不具有重复播放功能的。读者可以根据第 5 章的内容进行扩展学习。

要实现播放音乐的功能，需要如下几个步骤：

（1）在工程文件中，添加“winmm.lib”静态库文件及头文件，参见 5.4 节。

（2）实现 CTetrisView 类中的 PlayBackMusic()成员函数，其代码如代码 12.6 所示。

代码 12.6　CTetrisView 类的 PlayBackMusic 成员函数实现

```
01  #include <mmsystem.h>                           //插入系统 API 头文件
02  ...
03  void CTetrisView::PlayBackMusic(BOOL bCheck)
04  {
05      //指定文件并播放
06      if(bCheck)
07      {                                           //播放指定音乐文件
08          sndPlaySound("music.wav",SND_ASYNC);
09      }
10      else
11      {                                           //停止播放
12          sndPlaySound(NULL,SND_PURGE);
13      }
14  }
```

12.5.5　游戏等级设置对话框的实现

俄罗斯方块游戏的初始等级设置采用对话框方式实现，其步骤如下所述。

（1）创建一个对话框资源，并添加相应的控件资源，对话框中的资源 ID 及名称如表 12.13 所示。

表 12.13　游戏等级对话框资源ID及名称对照

ID	名　称
IDD_LEVEL_DLG	游戏等级设置对话框
IDC_LEVEL_EDIT	等级编辑框
IDOK	确定按钮
IDCANCEL	取消按钮

（2）游戏等级设置对话框中的资源放置，如图 12.10 所示。



图 12.10　游戏等级设置对话框

（3）给 setup.ini 配置文件增加一个小节用于记录设置游戏的等级，其格式如下：

```
...
[SETUP]
level=1;
```

（4）游戏等级设置对话框类的声明中，包含了两个函数。一个是单击“确定”按钮响应；另一个是单击“取消”按钮响应。其代码如代码 12.7 所示。

代码 12.7　游戏等级设置对话框类的声明

```
01  #if !defined(AFX_LEVELDLG_H__)
02  #define AFX_LEVELDLG_H__
03
04  // LevelDlg.h 游戏等级设置对话框类声明头文件
05  ////////////////////////////////////////
06  // CLevelDlg 对话框类声明
07
08  class CLevelDlg : public CDialog
09  {
10  public:
11      CLevelDlg(CWnd* pParent = NULL);        //构造函数
12
13      enum { IDD = IDD_LEVEL_DLG };           //资源加载
14      int     m_level;                        //游戏等级数据成员
15
16      protected:
```

```
17      virtual void DoDataExchange(CDataExchange* pDX);
18
19  protected:
20
21      virtual void OnOK();                       //单击“确定”按钮响应
22      virtual void OnCancel();                   //单击“取消”按钮响应
23      virtual BOOL OnInitDialog();               //初始化对话框
24      DECLARE_MESSAGE_MAP()
25  };
26
27  #endif
```

（5）游戏等级设置对话框通过初始化对话框时，把配置文件中的当前等级参数读出来并显示，然后根据用户单击“确定”按钮进行相应的写入操作。该对话框类的实现代码如代码 12.8 所示。

代码 12.8　游戏等级设置对话框类的实现

```
01  // LevelDlg.cpp 类实现源文件
02
03
04  #include "stdafx.h"                            //插入头文件
05  #include "Tetris.h"
06  #include "LevelDlg.h"                          //插入类声明头文件
07
08  ////////////////////////////////////////
09  // CLevelDlg 对话框类实现
10
11
12  CLevelDlg::CLevelDlg(CWnd* pParent /*=NULL*/)
13      : CDialog(CLevelDlg::IDD, pParent)
14  {
15      m_level = 0;                               //初始化游戏等级
16  }
17
18
19  void CLevelDlg::DoDataExchange(CDataExchange* pDX)
20  {
21      CDialog::DoDataExchange(pDX);
22                                                 //游戏等级设置变量与资源映射
23      DDX_Text(pDX, IDC_LEVEL_EDIT, m_level);
24      DDV_MinMaxInt(pDX, m_level, 1, 10);        //设置变量最大和最小值
25  }
26  BEGIN_MESSAGE_MAP(CLevelDlg, CDialog)
27  END_MESSAGE_MAP()
28
29  ///////////////////////////////////////////
30  // CLevelDlg 消息响应函数映射
31
32  void CLevelDlg::OnOK()
33  {
34      if(UpdateData(TRUE))                       //更新变量数据
35      {
36          CString tmp;                           //临时变量
37          tmp.Format("%d", m_level);             //转换成字符串
38                                                 //写入配置文件
39          WritePrivateProfileString("SETUP", "level", tmp, ".\\setup.ini");
```

```
40            CDialog::OnOK();                          //调用基类函数
41        }
42    }
43
44    void CLevelDlg::OnCancel()
45    {
46        CDialog::OnCancel();                          //调用基类退出函数
47    }
48
49    BOOL CLevelDlg::OnInitDialog()
50    {
51        CDialog::OnInitDialog();                      //调用基类初始化对话框函数
52
53        char pszTmp[128] = {0};                       //临时数组，用于存放字符串
54
55                                                      //读取配置文件中的初始等级
56        GetPrivateProfileString("SETUP", "level", "0", pszTmp, 127, ".\\setup.ini");
57
58        m_level = atoi(pszTmp);                       //转换字符串为数字
59
60        UpdateData(FALSE);                            //更新变量显示
61
62        return TRUE;                                  //返回 TRUE 初始化成功
63    }
```

代码解析：代码第 58 行是调用 atoi()函数，将读取出来的游戏等级字符串转换成数字，然后更新到屏幕上显示。

12.6　俄罗斯方块游戏的核心算法设计与实现

在前面的章节中已经讲解了俄罗斯方块游戏的菜单和各种对话框的实现。本节将对俄罗斯方块游戏的核心算法的设计和实现进行讲解。

12.6.1　主游戏类的设计

俄罗斯方块的主游戏类，负责显示游戏界面、方块、分数等级等内容，同时还要管理游戏的操作输入。

1. 游戏界面和方块的显示

（1）把游戏中的方块分为两类数组来保存，一类是将要出现的方块数组（Will）；另一类是当前已出现的方块数组（Now）。

（2）使用随机数生成函数，生成的随机数字并取余，得到当前生成方块类型值（1-7）。

（3）用方块类型值，去得到相应的方块样式，样式是已经定义好的 7 种。

（4）将当前生成的方块数据更新到将要显示的方块数组中。

（5）把游戏背景图片读入到内存中并显示出来。

（6）每次开始游戏时，将 Will 复制到 Now 数组中，并重新生成 Will 数组数据。

（7）使用定时器自动更新 Now 数组中的数据，即向下移动。

（8）把 Now 数组中的数据，转换并显示出来。

（9）判断是否与累积的方块重合，如果不是，则继续向下移动；如果是，则将 Will 数组中的数据复制到 Now 数组中。

（10）调用游戏规则类对象的接口函数，来判断当前游戏状态，并显示游戏分数及游戏等级。

2．游戏操作输入的处理

（1）在 CTetrisView 类中，添加接收玩家按下的方向键消息（OnKeyDown），并调用主游戏对象的接口函数 Move()将当前移动的方向数据传递过去。

（2）判断所按方向按键，当方向为“左”或者“右”时，判断是否碰壁。如果是，则不进行左右移动。如果不是，则把保存方块位置数组中的坐标数据进行相应的更新。当方向为“上”时，则调用旋转成员函数。

（3）向下移动，判断是否落到最下一行。如果不是，则把保存方块位置数组中的坐标数据进行相应更新。如果是，则继续判断是否填充这一行或者多行为完整一行或者多行；如果是，则把这一行或者多行进行消除；如果不是，则更新整个显示方块数组中的数据。

3．游戏计分的处理

游戏中每消除一行方块，程序就按照需求分析中的公式进行分数的累加及计算。

4．游戏升级的处理

（1）每消除一行，就调用游戏规则类的对象 Rule 的成员函数，对当前已经消除行数进行计算。

（2）当到达 30 层时，就自动将当前游戏等级增加 1。

（3）先关闭当前的 CTetrisView 类中的定时器。

（4）根据需求分析，对不同的等级的游戏速度进行设置。

（5）重新设置定时器的时钟。

12.6.2　主游戏类的实现

主游戏类的实现，可以分为如下几个步骤：

（1）声明主游戏类 CRussia，包含了消行处理、方块移动、方块旋转、碰撞判断、方块随机出现等成员函数。其代码如代码 12.9 所示。

代码 12.9　主游戏类 CRussia 的声明

```
01  #ifndef __RUSSIA_H__
02  #define __RUSSIA_H__
03
04  #include "Rule.h"                              //插入游戏规则类的声明
05
06  #define KEY_LEFT 1                             //定义按键方向宏
07  #define KEY_RIGHT 2
08  #define KEY_DOWN 3
09  #define KEY_UP 4
```

```
10
11  class CRussia                                  //声明 CRussia 类
12  {
13  public:
14      CRussia();                                 //构造函数
15      virtual ~CRussia();                        //析构函数
16      void LineDelete();                         //消行函数
17      void Move(int direction);                  //方块移动函数
                                                   //方块旋转
18      bool Change(int a[][4],CPoint p,int  b[][100]);
19      bool Meet(int a[][4],int direction,CPoint p); //判断碰撞
20      void DrawWill();                           //绘将要出现方块
21      void DrawBK(CDC*pDC);                      //绘界面
22      void DrawScore(CDC*pDC);                   //绘分数
23      void GameStart();                          //游戏开始
24      void HeroWrite();                          //英雄榜判断
25
26      int Russia[100][100];                      //游戏数组
27      int Now[4][4];                             //当前图形
28      int Will[4][4];                            //上一图形
29      int After[4][4];                           //变换后的图形
30      CPoint NowPosition;                        //当前图形的左上角位置
31      int Count;                                 //当前可能出现的图形形状数
32      bool end;                                  //游戏结束
33      int m_Level;                               //级别
34      int m_Speed;                               //速度
35      int m_Score;                               //分数
36      int m_CountLine;                           //合计消除行数
37      int m_RowCount,m_ColCount;                 //行列数
38      CBitmap fkMap;                             //方块
39      CBitmap bkMap;                             //界面
40      CRule rule;                                //游戏规则类对象
41  };
42  #endif
```

（2）实现 CRussia 类的基本功能函数，包括构造函数、析构函数及绘方块函数等。其代码如代码 12.10 所示。

代码 12.10　主游戏 CRussia 类的基本功能函数实现

```
01  #include "stdafx.h"
02  #include "Tetris.h"
03  #include "Russia.h"                        //插入类声明头文件
04  #include "HeroDlg.h"                       //插入英雄榜对话框类声明的头文件
05  /////////////////////////////////////////
06  //构造函数
07  ////////////////////////////////////////
08  CRussia::CRussia()
09  {
10      bkMap.LoadBitmap(IDB_BACK);            //加载背景图片
11      fkMap.LoadBitmap(IDB_FANGKUAI);        //加载方块图片
12  }
13  //析构函数
14  CRussia::~CRussia()
15  {
```

```
16  }
17  //绘方块图
18  void CRussia::DrawWill()
19  {
20      int i,j;
21      int k=4,l=4;
22
23      for(i=0;i<4;i++){                       //把将要出现数组赋值为零
24          for(j=0;j<4;j++){
25              Now[i][j]=Will[i][j];
26              Will[i][j]=0;
27          }
28      }
29      srand(GetTickCount());                  //初始化随机数种子
30      int nTemp=rand()%Count;                 //取余，得到方块样式
31
32      switch(nTemp)                           //根据随机数不同，生成不同的方块
33      {
34      case 0:
35          Will[0][0]=1;                       //方块样式列表 1
36          Will[0][1]=1;
37          Will[1][0]=1;
38          Will[1][1]=1;
39          break;
40      case 1:                                 //方块样式列表 2
41          Will[0][0]=1;
42          Will[0][1]=1;
43          Will[1][0]=1;
44          Will[2][0]=1;
45          break;
46      case 2:                                 //方块样式列表 3
47          Will[0][0]=1;
48          Will[0][1]=1;
49          Will[1][1]=1;
50          Will[2][1]=1;
51          break;
52      case 3:                                 //方块样式列表 4
53          Will[0][1]=1;
54          Will[1][0]=1;
55          Will[1][1]=1;
56          Will[2][0]=1;
57          break;
58      case 4:                                 //方块样式列表 5
59          Will[0][0]=1;
60          Will[1][0]=1;
61          Will[1][1]=1;
62          Will[2][1]=1;
63          break;
64      case 5:                                 //方块样式列表 6
65          Will[0][0]=1;
66          Will[1][0]=1;
67          Will[1][1]=1;
68          Will[2][0]=1;
69          break;
70      case 6:                                 //方块样式列表 7
71          Will[0][0]=1;
72          Will[1][0]=1;
73          Will[2][0]=1;
74          Will[3][0]=1;
```

```
75              break;
76          default:
77              break;
78          }
79
80          int tmp[4][4];                      //临时方块样式数组
81          for(i=0;i<4;i++)
82          {
83              for(j=0;j<4;j++)
84              {
85                  tmp[i][j]=Will[j][3-i]; //相当于把方块换一个方向
86              }
87          }
88
89          for(i=0;i<4;i++)
90          {
91              for(j=0;j<4;j++)
92              {
93                  if(tmp[i][j]==1)            //判断当前位置是否有填充
94                  {
95                      if(k>i) k=i;
96                      if(l>j) l=j;
97                  }
98              }
99          }
100
101         for(i=0;i<4;i++)
102         {
103             for(j=0;j<4;j++)
104             {
105                 Will[i][j]=0;               //清空出现方块数组
106             }
107         }
108
109         for(i=k;i<4;i++)                    //把变换后的矩阵移到左上角
110         {
111             for(j=l;j<4;j++)
112             {
113                 Will[i-k][j-l]=tmp[i][j];
114             }
115         }
116
117         NowPosition.x=0;                    //开始位置
118         NowPosition.y=m_ColCount/2;
119 }
```

代码解析：代码第 18 行的函数生成方块函数，其流程是利用随机数生成方块类型，再根据方块类型来决定创建的方块样式。这里有一个技巧：方块样式是先定义好，再由随机数进行选择。

（3）实现主游戏 CRussia 类中的界面显示和分数显示函数，其代码如代码 12.11 所示。

代码 12.11　绘界面及分数函数实现

```
01  //绘游戏界面
02  void CRussia::DrawBK(CDC*pDC)
03  {
04      CDC Dc;
05      if(Dc.CreateCompatibleDC(pDC)==FALSE)
```

```
06        {
07            AfxMessageBox("Can't create DC");
08        }
09
10       Dc.SelectObject(bkMap);                    //画背景
11        pDC->BitBlt(0,0,540,550,&Dc,0,0,SRCCOPY);
12        DrawScore(pDC);                            //画分数、速度、难度
13
14        for(int i=0;i<m_RowCount;i++)              //如果有方块，显示方块
15        {
16            for(int j=0;j<m_ColCount;j++)
17            {
18                if(Russia[i][j]==1)
19                {
20                    Dc.SelectObject(fkMap);       //设置图片对象
21                    pDC->BitBlt(j*30,i*30,30,30,&Dc,0,0,SRCCOPY);
22                }
23            }
24        }
25
26        for(int n=0;n<4;n++)                       //预先图形
27        {
28            for(int m=0;m<4;m++)
29            {
30                if(Will[n][m]==1)
31                {
32                    Dc.SelectObject(fkMap);
33                    pDC->BitBlt(365+m*30,240+n*30,30,30,&Dc,0,0,SRCCO
                      PY);
34                }
35            }
36        }
37    }
38    //绘分数和等级
39    void CRussia::DrawScore(CDC*pDC)
40    {
41        int nOldDC=pDC->SaveDC();
42
43        CFont font;
44                                                   //设置字体
45        if(0==font.CreatePointFont(300,"Comic Sans MS"))
46        {
47            AfxMessageBox("Can't Create Font");
48        }
49        pDC->SelectObject(&font);
50
51        CString str;
52        pDC->SetTextColor(RGB(39,244,10));        //设置字体颜色及其背景颜色
53        pDC->SetBkColor(RGB(255,255,0));
54
55        str.Format("%d",m_Level);
56        if(m_Level>=0)
57            pDC->TextOut(420,120,str);             //输出等级数字
58        str.Format("%d",m_CountLine);
59        if(m_Speed>=0)
60            pDC->TextOut(420,64,str);              //输出消除行数
61
62        str.Format("%d",m_Score);
63        if(m_Score>=0)
```

```
64          pDC->TextOut(420,2,str);            //输出分数
65      pDC->RestoreDC(nOldDC);
66  }
```

代码解析：代码第 14 行利用循环语句遍历方块全局数组，先绘制已经存在界面上的方块图像，然后再利用代码第 26 行的循环语句，将新生成的方块绘制出来。

（4）现在有了显示函数后，再给主游戏类添加游戏中方块操作函数，包含方块移动和方块旋转函数。其代码如代码 12.12 所示。

代码 12.12　主游戏类中的方块操作函数实现

```
01  ////////////////////////////////////////
02  //移动方块
03  ////////////////////////////////////////
04  void CRussia::Move(int direction)
05  {
06      if(end) return;                     //如果游戏结束，直接返回
07
08      switch(direction)                   //根据方向进行移动
09      {
10          case KEY_LEFT:                  //向左
11          if(Meet(Now,KEY_LEFT,NowPosition)){
12                  break;}
13          NowPosition.y--;                //Y 坐标减少
14          break;
15      case KEY_RIGHT:                     //向右
16          if(Meet(Now,KEY_RIGHT,NowPosition)){
17                  break;}
18          NowPosition.y++;                //Y 坐标增加
19          break;
20      case KEY_DOWN:                      //向下
21          if(Meet(Now,KEY_DOWN,NowPosition))
22          {
24              LineDelete();               //消除行
25              break;
26          }
27          NowPosition.x++;                //X 坐标增加
28          break;
30      case KEY_UP:                        //向上
31          Meet(Now,KEY_UP,NowPosition);   //判断方块是否可以移动或者转动
32          break;
33      default:
34          break;
35      }
36  }
37  ///////////////////////////////////////////////
38  //方块旋转
39  ///////////////////////////////////////////////
40  bool CRussia::Change(int a[][4],CPoint p,int  b[][100])
41  {
42      int tmp[4][4];                      //临时方块存放数组
43      int i,j;
44      int k=4,l=4;
45      for(i=0;i<4;i++)
46      {
47          for(j=0;j<4;j++)
48          {
```

```
49                  tmp[i][j]=a[j][3-i];
50                  After[i][j]=0;                  //存放变换后的方块矩阵
51              }
52          }
53
54          for(i=0;i<4;i++)
55          {
56              for(j=0;j<4;j++)
57              {
58                  if(tmp[i][j]==1)                //判断当前位置是否有填充
59                  {
60                      if(k>i) k=i;
61                      if(l>j) l=j;
62                  }
63              }
64          }
65          for(i=k;i<4;i++)
66          {
67              for(j=l;j<4;j++)
68              {
69                  After[i-k][j-l]=tmp[i][j];  //把变换后的矩阵移到左上角
70              }
71          }
72
73          for(i=0;i<4;i++)                        //判断是否接触，如果是，则返回失败
74          {
75              for(j=0;j<4;j++)
76              {
77                  if(After[i][j]==0)
78                  {
79                      continue;                   //重新查找
80                  }
81                  if(((p.x+i)>=m_RowCount)||((p.y+j)<0)||((p.y+j)>=m_ColC
                    ount))
82                  {
83                      return false;               //返回旋转失败
84                  }
85                  if(b[p.x+i][p.y+j]==1)
86                  {
87                      return false;               //返回旋转失败
88                  }
89              }
90          }
91          return true;                            //可以旋转
92      }
```

代码解析：代码第 40 行的 Change()函数主要是根据 Now 数组中的数据，进行转换并显示到主游戏界面上，然后再将其显示到左上角，最后判断旋转后的方块会不会碰到边界或者其他方块的情况。如果不会就可以旋转，否则不能进行旋转。

（5）在移动时进行碰撞判断。碰撞函数的实现如代码 12.13 所示。

代码 12.13　碰撞函数实现

```
01  ///////////////////////////////////////////////////
02  //判碰撞，遇到了边界或者有其他方块挡住
03  ///////////////////////////////////////////////////
04  bool CRussia::Meet(int a[][4],int direction,CPoint p)
05  {
```

```
06        int i,j;
07        //先把原位置清零
08        for(i=0;i<4;i++)
09        {
10            for(j=0;j<4;j++)
11            {
12                if(a[i][j]==1)
13                {
14                    Russia[p.x+i][p.y+j]=0;     //清空数据
15                }
16            }
17        }
18
19        for(i=0;i<4;i++)                      //循环遍历数组
20        {
21            for(j=0;j<4;j++)
22            {
23                if(a[i][j]==1)
24                {
25                    switch(direction)
26                    {
27                    case KEY_LEFT:            //向左移动
28                        if((p.y+j-1)<0) goto exit;
29                        if(Russia[p.x+i][p.y+j-1]==1) goto exit;
30                        break;
31                    case KEY_RIGHT:           //向右移动
32                        if((p.y+j+1)>=m_ColCount) goto exit;
33                        if(Russia[p.x+i][p.y+j+1]==1) goto exit;
34                        break;
35                    case KEY_DOWN:            //向下移动
36                        if((p.x+i+1)>=m_RowCount) goto exit;
37                        if(Russia[p.x+i+1][p.y+j]==1) goto exit;
38                        break;
39                    case KEY_UP:              //向上，默认为变换
40                        if(!Change(a,p,Russia)) goto exit;
41                        for(i=0;i<4;i++)
42                        {
43                            for(j=0;j<4;j++)  //将现在的方块样式进行变化
44                            {
45                                Now[i][j]=After[i][j];
46                                a[i][j]=Now[i][j];
47                            }
48                        }
49                        return false;
50                        break;
51                    }
52                }
53            }
54        }
55
56        int x,y;
57        x=p.x;                                //得到当前点的 X 坐标
58        y=p.y;                                //得到当前点的 Y 坐标
59
60        switch(direction)                     //移动位置，重新给数组赋值
61        {
62        case 1:
63            y--;break;                        //减少 Y 坐标
64        case 2:
```

```
65          y++;break;                          //增加 Y 坐标
66      case 3:
67          x++;break;                          //增加 X 坐标
68      case 4:
69          break;                              //向上时，不进行操作
70      }
71      for(i=0;i<4;i++)
72      {
73          for(j=0;j<4;j++)
74          {
75              if(a[i][j]==1)                  //根据标志进行填充
76              {
77                  Russia[x+i][y+j]=1;
78              }
79          }
80      }
81
82      return false;                           //没有碰撞发生
83  exit:
84      for(i=0;i<4;i++)
85      {
86          for(j=0;j<4;j++)
87          {
88              if(a[i][j]==1)
89              {
90                  Russia[p.x+i][p.y+j]=1;
91              }
92          }
93      }
94      return true;                            //有碰撞发生
95  }
```

代码解析：代码第 27、31、35、39 行的条件是根据当前移动的方向来判断移动后的数据是否与全局方块数组中的数据重复，只要不重复，就可以移动。否则，重新恢复到原来的样子。

（6）游戏中要得分，就必须消除满行的方块。在实现方块行的消除函数时，要注意，有 4 种不同类型的消除行，是需要分别进行处理的。其代码如代码 12.14 所示。

代码 12.14　消行处理函数的实现

```
01  ////////////////////////////////////////
02  //行消除函数
03  ///////////////////////////////////////
04  void CRussia::LineDelete()
05  {
06      int m=0;                                //本次共消除的行数
07      bool flag=0;
08      for(int i=0;i<m_RowCount;i++)
09      {                                       //检查是否要消行
10          flag=true;
11          for(int j=0;j<m_ColCount;j++)
12          {
13              if(Russia[i][j]==0)             //判断每一列
14              {
15                  flag=false;
16              }
17          }
```

```
18          if(flag==true)                      //如果要消行
19          {
20              m++;
21              for(int k=i;k>0;k--)
22              {                               //上行给下行
23                  for(int l=0;l<m_ColCount;l++)
24                  {
25                      Russia[k][l]=Russia[k-1][l];
26                  }
27              }
28
29              for(int l=0;l<m_ColCount;l++)   //第一行为 0
30              {
31                  Russia[0][l]=0;
32              }
33          }
34      }
35
36      DrawWill();                             //画准备方块
37
38      switch(m)                               //消行判断，是 1～4 行中那个
39      {
40      case 1:                                 //消除 1 行
41          m_Score= m_Score + 10 + m_Level * 10;
42          break;
43      case 2:                                 //消除 2 行
44          m_Score= m_Score + 30 + m_Level * 10;
45          break;
46      case 3:                                 //消除 3 行
47          m_Score= m_Score + 50 + m_Level * 10;
48          break;
49      case 4:                                 //消除 4 行
50          m_Score= m_Score + 100 + m_Level * 10;
51          break;
52      default:
53          break;
54      }
55
56      m_CountLine+=m;                         //累积消除方块行数
57
58      m_Level = rule.UpLevel(m_CountLine);    //升级判断
59
60      end = rule.Win(Now, Russia, NowPosition);   //游戏结束判断
61
62      m_Speed=320 - m_Level * 20;                 //调整速度
63
64      if(end)                                     //判断结束标志
65      {
66          AfxMessageBox("游戏结束！");
67      }
68  }
```

代码解析；代码第 8 行，循环遍历全局方块数组，计算有几行可以进行消除，然后将相应的行从上向下移动。再根据消除不同的行数，进行计分。最后根据总消除的行数判断游戏是否已经可以升级。

（7）实现主游戏类的外部控制接口函数包括：游戏开始及英雄榜处理函数，其代码如代码 12.15 所示。

代码 12.15　外部控制接口函数实现

```
01  ////////////////////////////////////////
02  //游戏开始
03  ///////////////////////////////////////
04  void CRussia::GameStart()
05  {
06      end=false;                          //运行结束标志
07     m_Score=0;                           //初始分数
08      m_RowCount=18;                      //行数
09      m_ColCount=12;                      //列数
10      Count=7;                            //方块种类
11      m_CountLine = 0;                    //合计消除行数为 0
12
13      char pszTmp[128] = {0};
14                                          //读取当前游戏等级
15      GetPrivateProfileString("SETUP", "level", "1",
16              pszTmp, 127, ".\\setup.ini");
17
18      m_Level = atoi(pszTmp);             //初始等级
19      m_Speed=320 - m_Level * 20;         //初始速度
20      rule.SetLevel(m_Level);
21
22      for(int i=0;i<m_RowCount;i++)       //清空界面数组
23      {
24          for(int j=0;j<m_ColCount;j++)
25          {
26              Russia[i][j]=0;
27          }
28      }
29      for(i=0;i<4;i++)                    //清空准备方块和当前方块数组
30      {
31          for(int j=0;j<4;j++)
32          {
33              Now[i][j]=0;
34              Will[i][j]=0;
35          }
36      }
37      //开始时将要出现方块没有生成，其不能赋值给当前方块数组，所以连续调用两次
38      DrawWill();
39      DrawWill();
40  }
41  ////////////////////////////////////////////////////////////////////
42  //英雄榜对话框弹出判断
43  ////////////////////////////////////////////////////////////////////
44  void CRussia::HeroWrite()
45  {
46      CHeroDlg dlg;                       //创建英雄榜对话框对象
47
48      char pszTmp[128] = {0};
49
50      int nHighScore = 0;                 //存储最高分临时变量
51                                          //读配置文件
52      GetPrivateProfileString("HERO", "score", "0",
53          pszTmp, 127, ".\\hero.ini");
54
55      nHighScore = atoi(pszTmp);          //转换字符串为数字
56
```

```
57      if(m_Score>nHighScore)              //比较最高分和当前分数
58      {
59
60          dlg.SetWriteFlg(TRUE);          //设置可写入标志
61
62          dlg.m_level = m_Level;          //设置等级
63
64          dlg.m_score = m_Score;          //设置分数
65
66          dlg.DoModal();                  //弹出对话框
67      }
68      else
69      {
70          AfxMessageBox("游戏结束,您未能进入英雄榜!");
71      }
72  }
```

12.6.3　游戏规则类的设计

游戏规则类，包含两个功能函数。

1．游戏胜负判断处理

（1）对当前游戏界面数组与当前方块数组进行比较，如果超过上边界，说明游戏以失败而结束。

（2）对当前等级进行比较，当游戏等级超过 10 级，则判断完全通关，游戏结束。

2．游戏升级处理

得到已消掉的方块行数，如果是 30 的整数倍时，说明已经可以升级。游戏升级算法如下：

```
游戏等级 = 当前已消除行数 / 30 + 当前游戏等级
```

12.6.4　游戏规则类的实现

游戏规则类的实现，包括如下几个步骤：

（1）声明游戏规则类，其中包含构造函数、析构函数、升级判断函数、胜负判断函数。其代码如代码 12.16 所示。

代码 12.16　游戏规则类的声明

```
01  #ifndef __RULE_H__
02  #define __RULE_H__
03
04  class CRule
05  {
06  public:
07      CRule();
08      ~CRule();
09      void SetLevel(int nLevel);          //设置当前等级
```

```
10      int UpLevel(int nLine);                  //升级判断
11      bool Win(int Now[4][4], int Russia [100][100], CPoint NowPosition);
                                                 //胜负判断
12  private:
13      int m_nLevel;                            //当前等级
14  };
15
16  #endif
```

（2）实现游戏规则类中的升级判断函数和胜负判断函数，其代码如代码 12.17 所示。

代码 12.17　游戏规则类的实现

```
01  #include "stdafx.h"                     //插入工程头文件
02  #include "Rule.h"                       //插入类声明头文件
03
04  CRule::CRule()                          //构造函数
05  {
06  }
07
08  CRule::~CRule()                         //析构函数
09  {
10  }
11
12  void CRule::SetLevel(int nLevel)        //设置当前游戏等级
13  {
14      m_nLevel = nLevel;
15  }
16
17  int CRule::UpLevel(int nLine)           //升级判断接口函数
18  {
19      if(nLine / 30)
20      {                                   //如果可以整除，等级升级
21          m_nLevel++;
22      }
23      return m_nLevel;                    //返回当前游戏等级
24  }
25
26  bool CRule::Win(int Now[4][4], int Russia [100][100], CPoint
    NowPosition)
27  {
28      if(m_nLevel == 11)                  //游戏等级超过最高
29      {                                   //消除行数已经超过 10 级，游戏结束
30          return true;
31      }
32
33      for(int i=0;i<4;i++)
34      {
35          for(int j=0;j<4;j++)
36          {
37              if(Now[i][j]==1)
38              {//到了顶点
39                  if(Russia[i+NowPosition.x][j+NowPosition.y]==1)
40                  {
41                      return true;            //游戏结束
42                  }
43              }
44          }
45      }
```

```
46
47        return false;                     //游戏未结束
48    }
```

代码解析：第 39 行判断当前的移动的方块坐标是否已经到达游戏界面的顶点，即竖坐标已经超过整个全局数组的位置，则返回真，表示游戏已经结束。

12.7　俄罗斯方块游戏的整合测试

在这一节中，笔者将根据前面的测试用例文档来进行俄罗斯方块游戏的整合测试。在这里，笔者只对其中几个主要的测试用例进行演示。

12.7.1　主菜单和界面显示功能测试的演示

这个测试用例主要是测试游戏的菜单和界面显示是否成功，根据测试用例文档，其测试步骤如下所述。

（1）运行俄罗斯方块程序，如图 12.11 所示。

（2）程序启动后，界面如图 12.12 所示。



图 12.11　运行俄罗斯方块程序

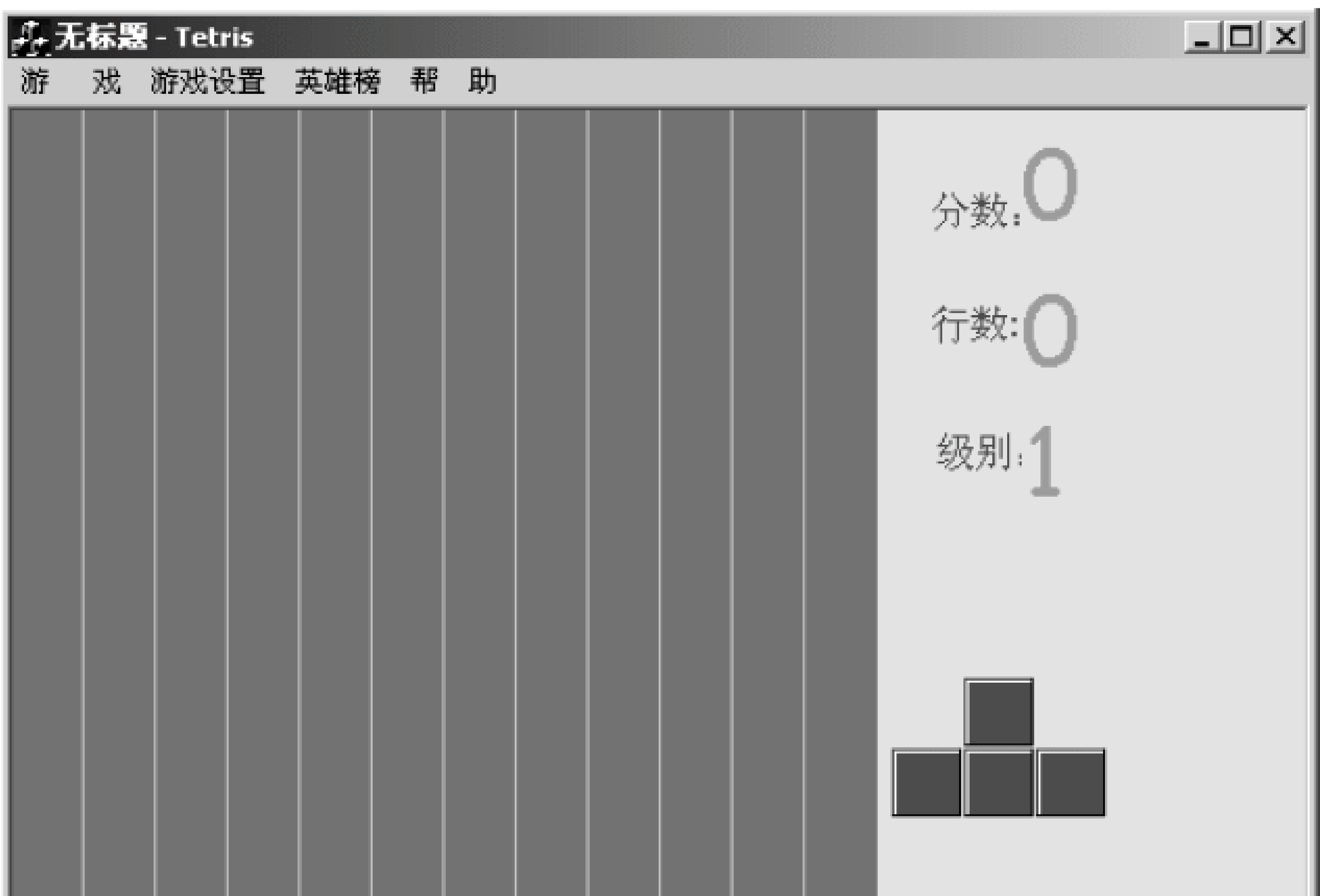
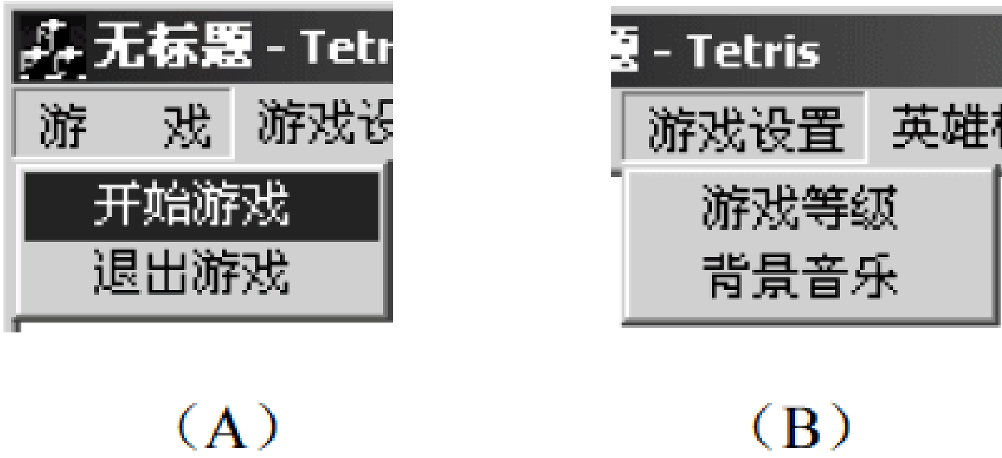


图 12.12　俄罗斯方块游戏界面

（3）分别选择主菜单中的“游戏”菜单栏（A）和“游戏设置”菜单栏（B），如图 12.13 所示。

判断结果：游戏的菜单和界面显示成功。填写测试用例编号 1.7.1 结果为“通过”。



（A）　（B）

图 12.13　俄罗斯方块游戏菜单

12.7.2　游戏等级选择功能测试的演示

游戏等级选择功能测试，根据测试用例文档，其测试步骤如下所述。

（1）选择“游戏设置”|“游戏等级”命令，如图 12.14 所示。

（2）在弹出的对话框中，指定当前游戏等级并单击“确定”按钮，如图 12.15 所示。

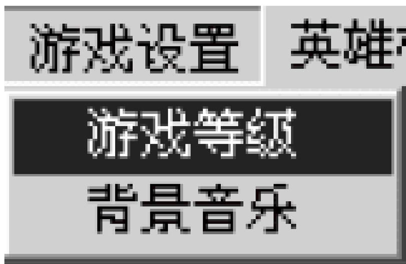


图 12.14　选择“游戏设置”|“游戏等级”命令



图 12.15　弹出“游戏等级设置”对话框

（3）查看主界面中，游戏提示的当前等级，如图 12.16 所示。

判断结果：游戏等级设置成功。填写测试用例编号 1.7.4 结果为“通过”。

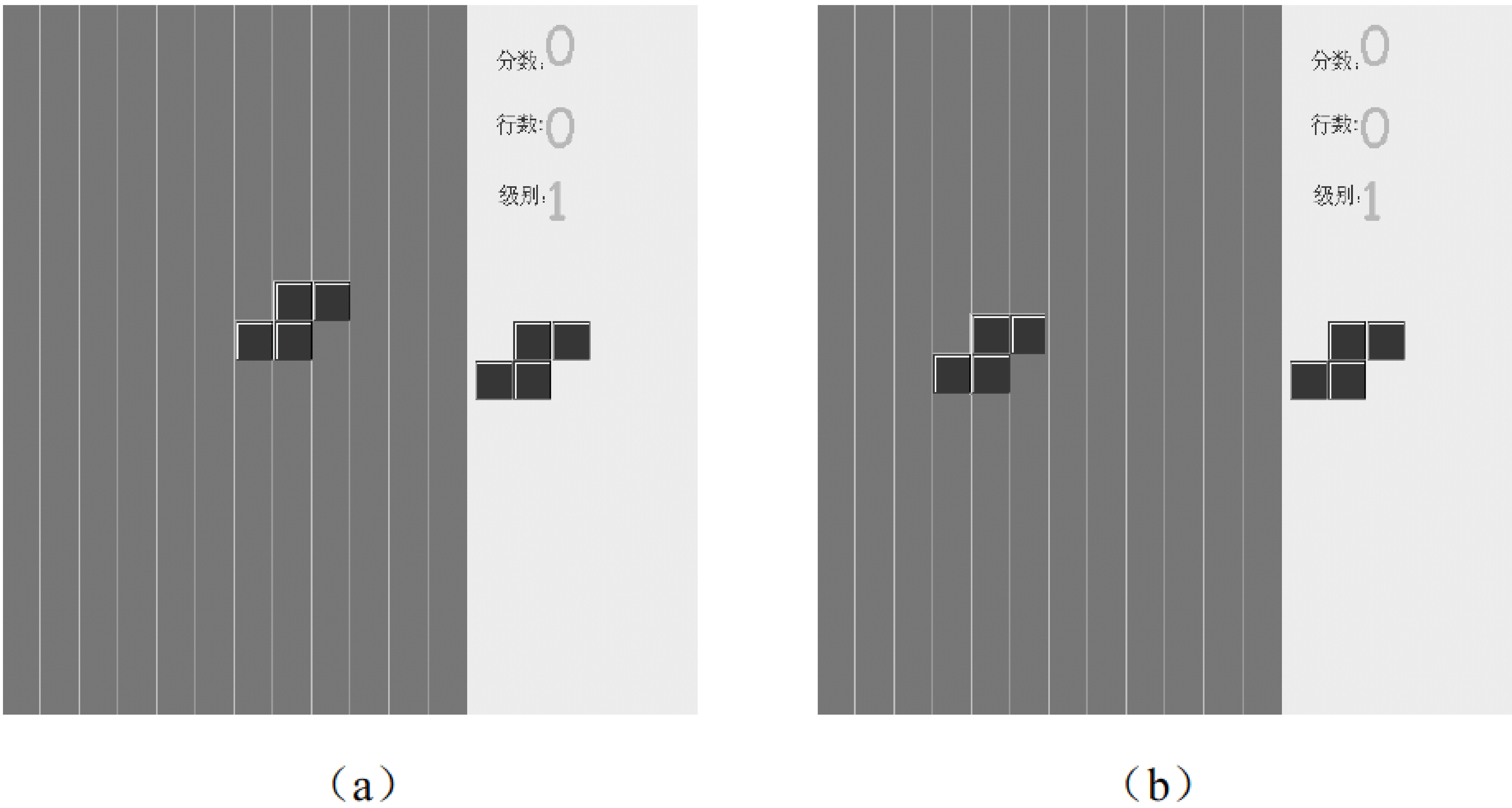


图 12.16　游戏提示当前游戏等级

12.7.3　方块移动功能测试的演示

测试游戏中的方块是否能够按照正确的方向移动并旋转（这里只演示其中的两个方向）。根据测试用例文档，其测试步骤如下。

（1）单击“左”按键，移动前如图 12.17（a）所示，移动后如图 12.17（b）所示。查看方块是否向“左”移动。



（a）　　（b）

图 12.17　方块向左移动

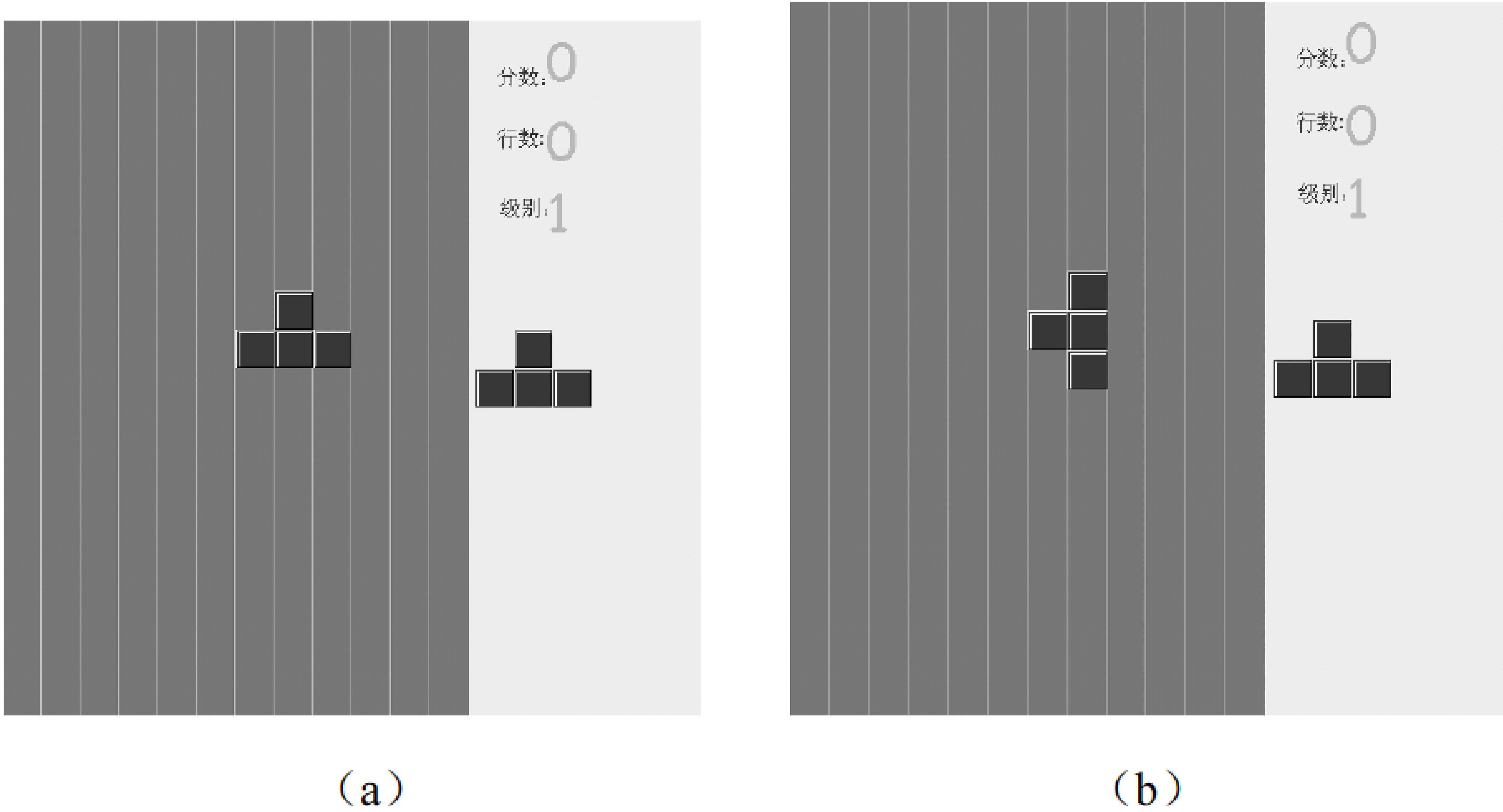
（2）单击“上”按键时，旋转前如图 12.18（a）所示，移动后如图 12.18（b）所示。查看方块是否进行旋转变化。

判断结果：游戏中的方块能够按照正确的方向移动并旋转。填写测试用例编号 1.7.3 结果为“通过”。

12.7.4　游戏规则功能测试的演示

测试俄罗斯方块游戏能否对结束和消行规则进行判断，根据测试用例文档，其测试步骤如下所述。

（1）运行游戏，并填充完一行方块，填充前如图 12.19（a）所示，填充后如图 12.19（b）

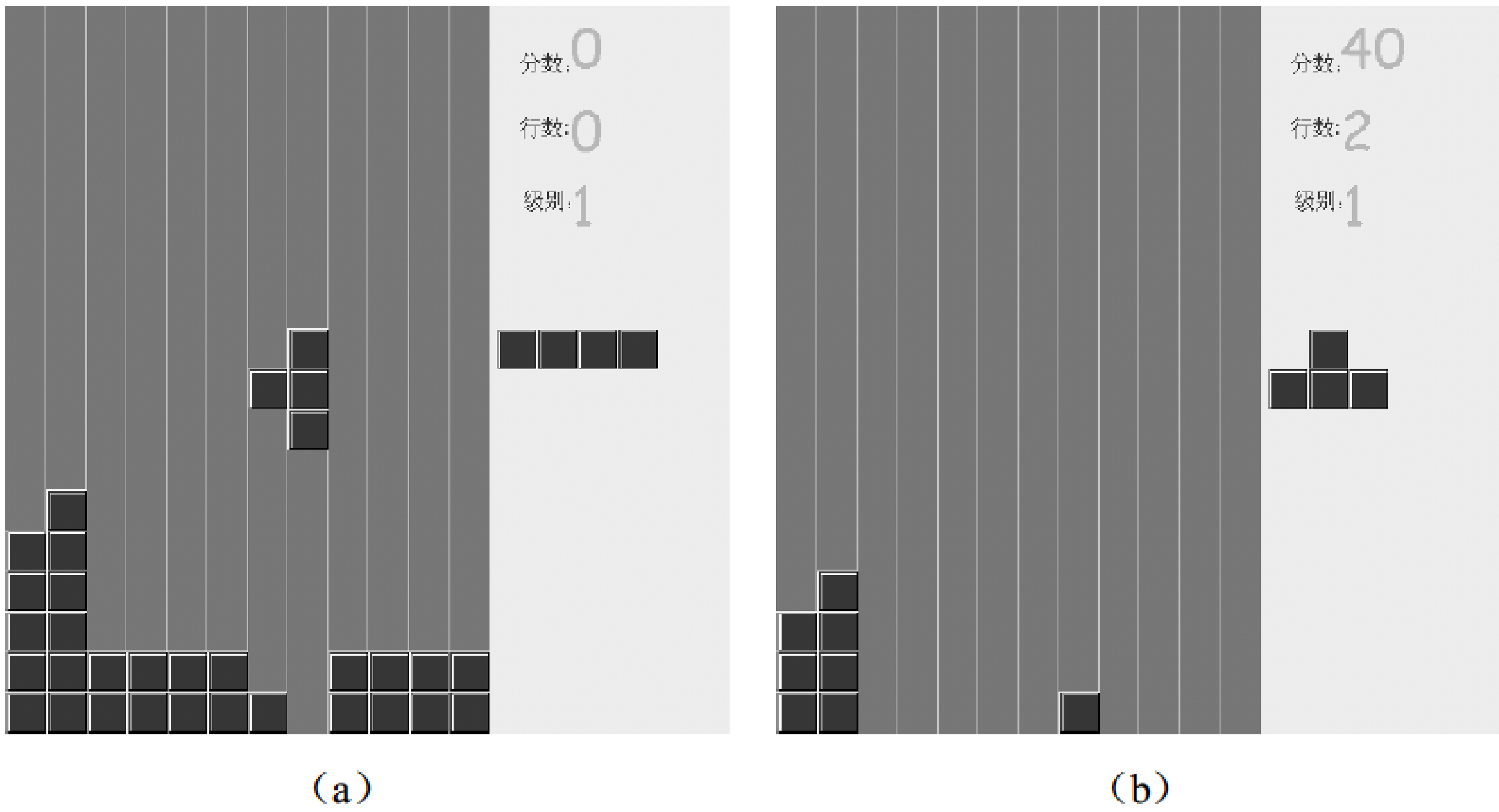


（a）　　　　　　　　　　（b）

图 12.18　方块旋转变化

所示。查看是否消除一行。

（2）让方块累积到上边界，查看是否有游戏结束提示，如图 12.20 所示。



（a）　　　　　　　　　　（b）

图 12.19　消行规则

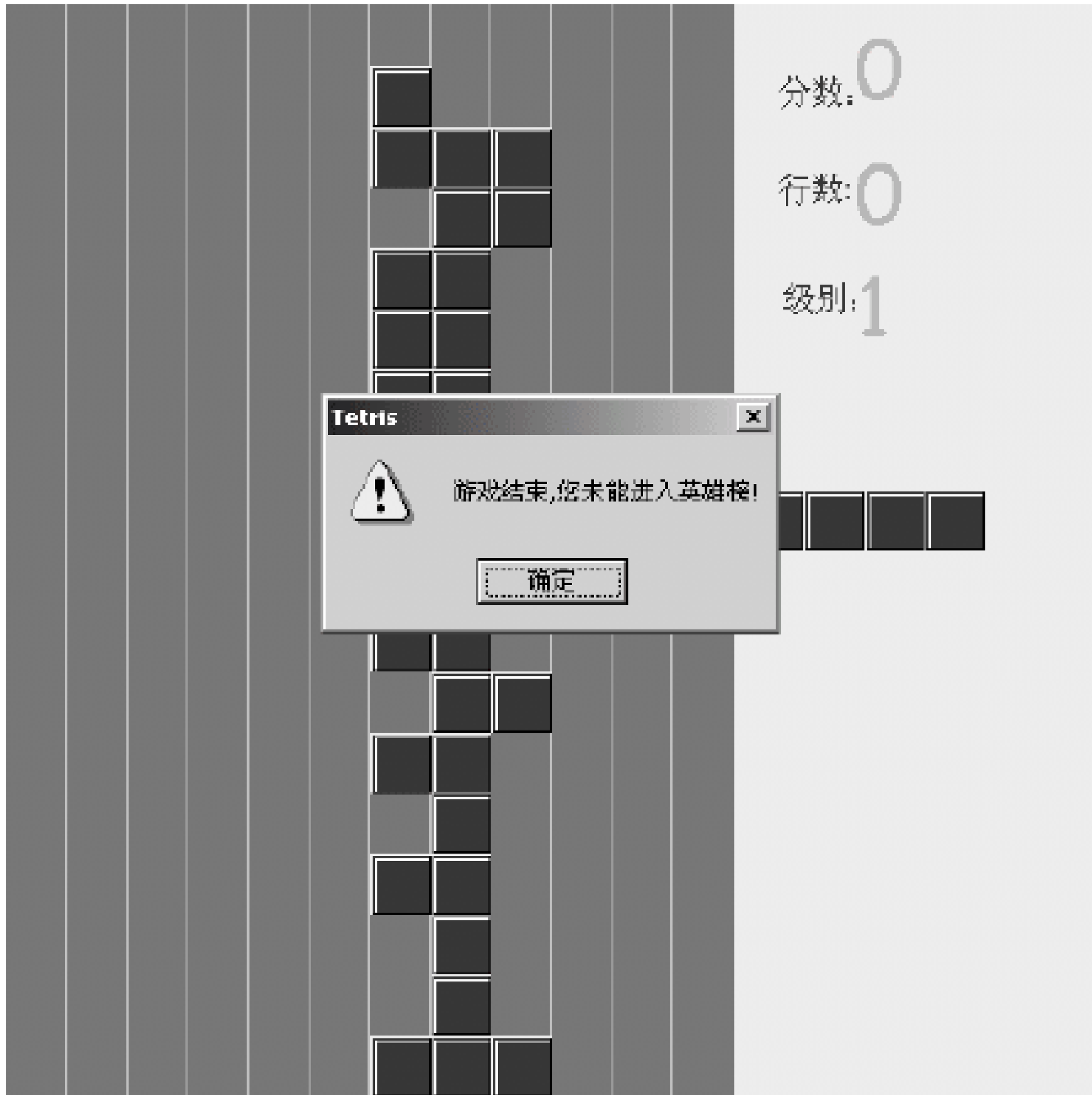


图 12.20　游戏结束提示

判断结果：俄罗斯方块游戏能够对结束和消行规则进行判断。填写测试用例编号 1.7.5 结果为“通过”。

12.7.5　游戏帮助功能测试的演示

测试俄罗斯方块游戏是否有帮助提示功能。根据测试用例文档，其测试步骤如下所述。

（1）选择“帮助”|“帮助”命令，如图 12.21 所示。

（2）游戏中弹出帮助对话框，如图 12.22 所示。

判断结果：俄罗斯方块游戏中的游戏帮助功能是正确的。填写测试用例编号 1.7.7 结果为“通过”。

12.7.6　游戏计分功能测试的演示

测试游戏中的计分功能是否正确，根据测试用例文档，其测试步骤如下所述。

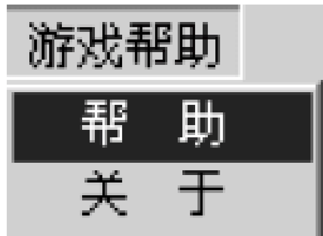


图 12.21　选择“游戏帮助”|“帮助”命令

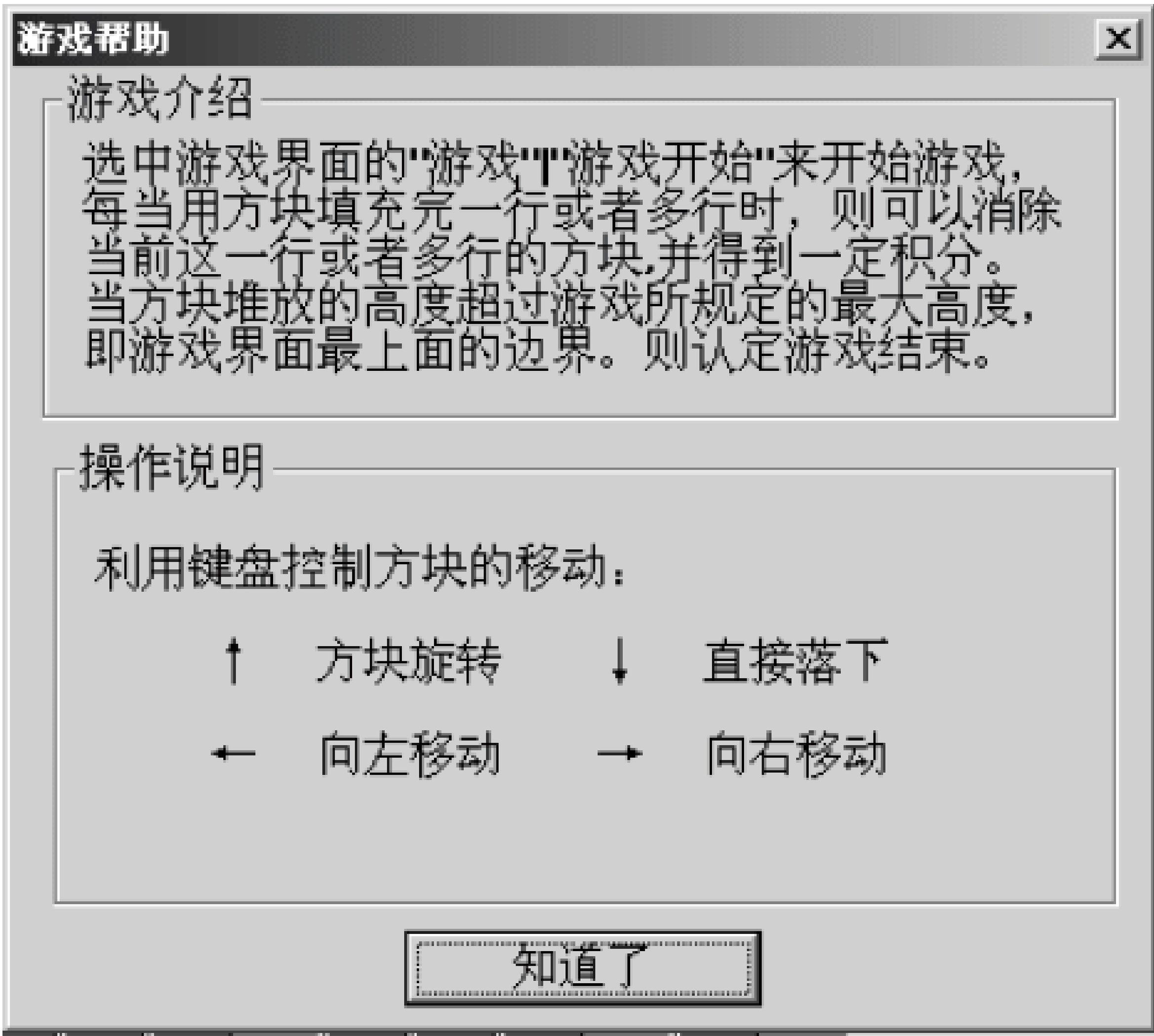


图 12.22　弹出“游戏帮助”对话框

（1）一次性消除“1”行方块，查看当前游戏得分，如图 12.23 所示。

（2）一次性消除“4”行方块，查看当前游戏得分，如图 12.24 所示。

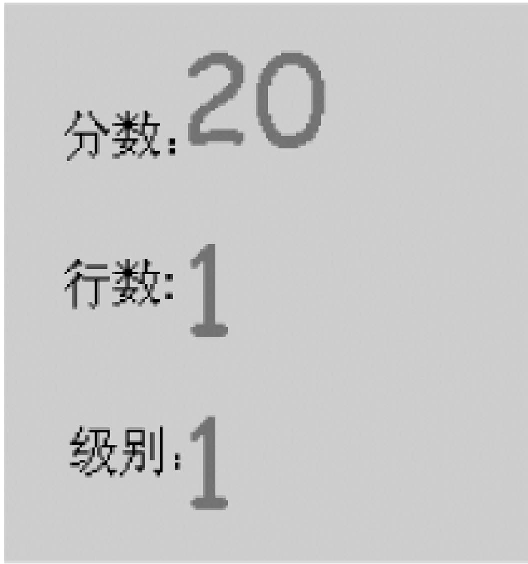


图 12.23　一次性消除“1”行方块得分提示



图 12.24　一次性消除“4”行方块得分提示

判断结果：俄罗斯方块游戏中计分功能是正确的。填写测试用例编号 1.7.10 结果为“通过”。

12.8　总　　结

通过本章俄罗斯方块游戏实例项目开发的学习，希望各位读者能够更加深入地掌握好游戏项目开发中各种文档的格式及编写方法。同时，知道自己如何开发一款俄罗斯方块游戏。本游戏最核心算法包括：游戏方块的移动、旋转、生成及消除等。这是这款游戏最难理解的部分，请读者采用一边阅读一边实践的方式来学习，这样才能深刻理解整个算法的核心思想。

第 13 章　连连看游戏项目开发

学习完第 12 章的俄罗斯方块游戏的开发，本章将继续介绍连连看游戏项目的开发。本章的内容结构和第 12 章大致相同，主要是为了帮助读者继续增加项目实际开发的经验，提高对各种文档的编写能力。

本章主要涉及的内容如下：

- 连连看项目的需求分析。
- 连连看游戏的概要设计。
- 连连看游戏操作界面设计。
- 连连看游戏的详细设计及代码。
- 连连看游戏的测试用例文档的编写及测试演示。

13.1　连连看游戏项目的需求分析

获得用户需求并对其进行详细分析，是项目开始的基础。只有获得明确的需求，并做出好的需求分析文档得到客户的认可，才能保证项目的成功。

技巧： 需求分析要尽量从最后使用者处获得，这样才能达到目的。

13.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到连连看游戏开发的资料如下所述。

1．连连看游戏概述

不管在哪个小游戏网站，“连连看”游戏总是排在最受玩家欢迎的排名的前几位。因为它是不分男女老少，适合大众的集休闲、趣味、益智和娱乐于一体的经典小游戏。

该游戏速度节奏快，画面清晰可爱，适合以 MM 为主体的细心的玩家。游戏中多样式的地图，使玩家在各个游戏水平都可以寻找到挑战的目标，长期地保持游戏的新鲜感。游戏增加了时间聘用制功能，让玩家更有挑战性和追求极速的欲望。

2．连连看的操作方法

第一次使用鼠标单击棋盘中的棋子，该棋子此时为“被选中”状态，以特殊方式显示；再次单击其他棋子，若该棋子与被选中的棋子图案相同，且把第一个棋子到第二个棋子连起来，中间的直线不超过 3 根，就消掉这一对棋子，否则第一个棋子恢复成未被选中状态，

而第二个棋子变成被选中状态。

3．连连看游戏的基本规则

每消去一对棋子，游戏限制时间会自动增加。当时间全部消耗完时，游戏以失败结束。在有限时间内，消除全部的棋子，自动升级，并重新开始新一个等级的游戏。每升一个等级，游戏的时间消耗就变得更快。

游戏中还可以利用快捷键来进行变化棋盘（3 次机会）和提示功能。

（1）变盘即对于未消除棋子重新排列。

（2）提示即对可以消除的方块进行提示。

4．英雄榜的显示及更新

游戏结束时，当有玩家的当前等级超过记录文件中的等级时，就弹出英雄榜对话框。要求玩家输入姓名，并记录在记录文件中。

5．游戏选择播放背景音乐

在游戏开始后，可以选择播放背景音乐。

6．游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

13.1.2　对语言描述进行需求分析

根据《连连看用户需求描述文档》中的内容，现在需要对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

技巧：在编写需求分析时，让编程人员参与更能够对需求描述功能化。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类连连看游戏。特制定本说明书用于描述某公司连连看项目开发的功能性需求。

2．文档范围

包含某公司连连看游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司连连看游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

《连连看用户需求描述文档》。

5．游戏具有的功能

5.1　能够显示主菜单和界面

游戏需要提供主菜单让玩家进行游戏设置，同时能够显示剩余时间、当前游戏等级等

相关信息到界面上。

5.2　能够实现时间限制功能

能够根据游戏状态自动增加或者减少当前时间限制长短。

5.3　能够根据规则消除相同棋子

游戏以鼠标进行操作，第一次单击棋盘中的棋子，该棋子此时为“被选中”状态，以特殊方式显示出来；再次单击其他棋子，若该棋子与被选中的棋子图案相同，且把第一个棋子到第二个棋子连起来，中间的直线不超过 3 根，则消掉这一对棋子。并且当前时间限制长短增加，否则第一个棋子恢复成未被选中状态，而第二个棋子变成被选中状态。当游戏时间全部消耗完时，则游戏结束。

5.4　游戏升级功能

当游戏中的棋子全部消除完毕时，游戏等级上升一个等级。游戏等级每增加一个等级，时间限制长度如表 13.1 所示。

表 13.1　游戏等级与时间限制

游 戏 等 级	时间限制长度	游 戏 等 级	时间限制长度
1	120s	6	70s
2	110s	7	60s
3	100s	8	50s
4	90s	9	40s
5	80s	10	30s

5.5　消除提示功能

在游戏中，如果玩家觉得自己无法找到一对消除棋子时，可以使用快捷键（F5）调出游戏中的提示功能，来提醒当前可消除的棋子。最多可以使用 3 次。

5.6　棋子换盘功能

当游戏中的棋子都已经全部无法消除时，可以使用快捷键（F6）调用棋子换盘功能，重新把棋子随机排列来继续游戏，该功能最多可以使用 3 次。

5.7　英雄榜的更新

当有玩家得到的等级超过当前记录等级，在结束游戏时，要求玩家把名字输入并保存下来。游戏初始时记录分数线为 1 级。

例如，第一个玩家的等级为 2 级，结束游戏时，那么这个玩家的等级将被保存下来并作为最高游戏等级记录。直到有玩家的等级超过 2 级，才能更新当前记录等级并在退出游戏时保存玩家名字及游戏等级。

5.8　游戏支持背景音乐功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.9　游戏提供帮助说明文档

在游戏菜单中，提供一个使用说明项，以方便对本游戏不了解的玩家对游戏进行操作和使用。

13.2　连连看游戏项目概要设计

根据需求分析文档，笔者编写的连连看游戏的概要设计文档内容如下所述。

技巧：概要设计时，功能划分以需要分析来扩展，会更方便和高效。

1．引言

为了让每个开发人员明白连连看游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

2．术语

3．参考文献

《连连看游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高鼠标及键盘操作的易用性。

4.2　需求概述

参见《连连看游戏需求分析说明书》。

5．总体设计

5.1　连连看游戏的功能架构（如图 13.1 所示）

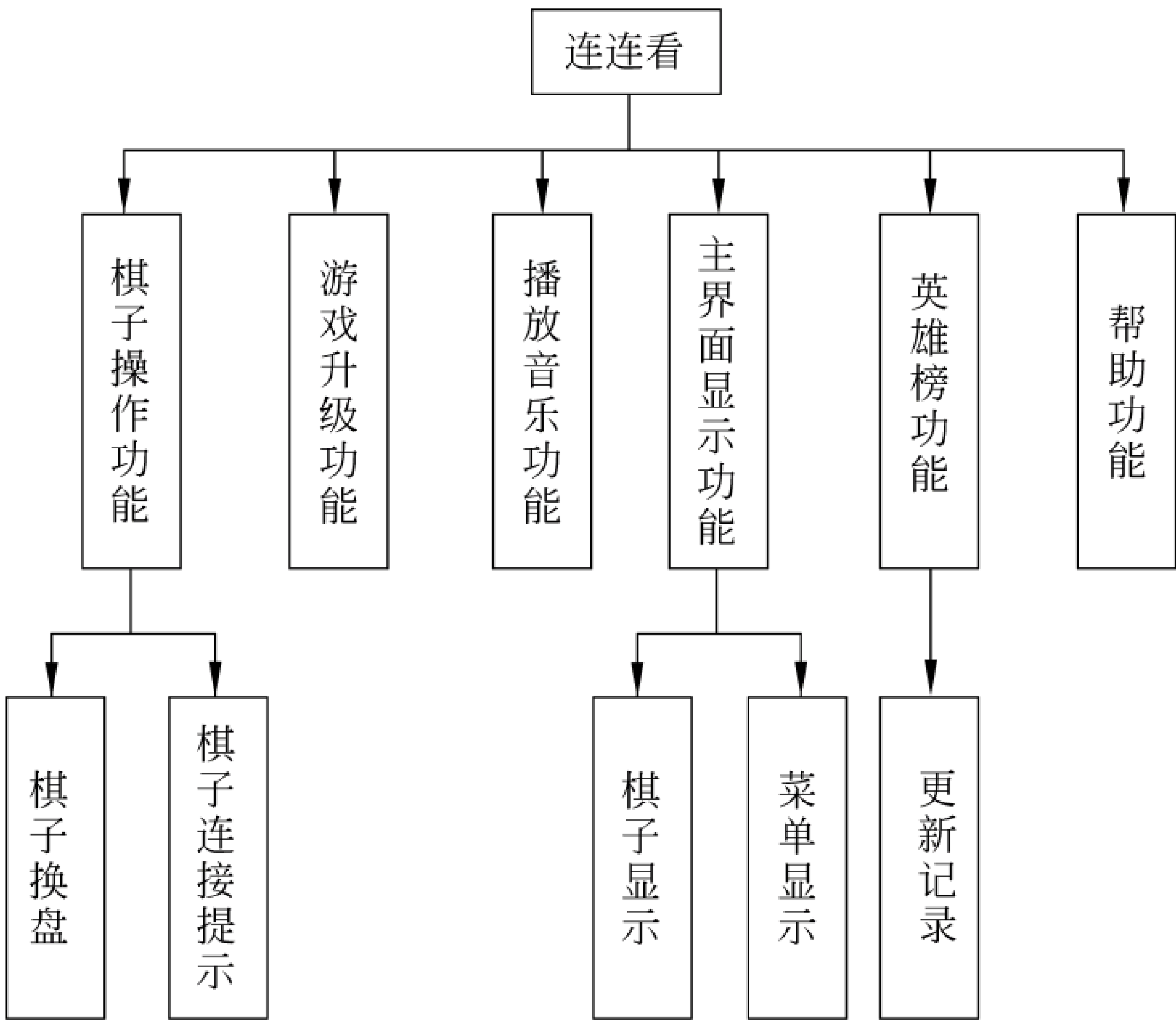


图 13.1　连连看功能架构

5.2　各功能处理流程

内容参见《连连看游戏各功能详细设计文档》。

6．接口设计

内容参见《连连看游戏操作界面设计文档》。

7．类结构设计

游戏由 6 个类组成，如图 13.2 所示。

- ❑ 主界面对话框类：主要负责主界面及菜单的显示、棋子消除、消除提示及换盘操作，同时还要负责时间控制等。
- ❑ 棋子类：主要负责棋子的选中，配对及查找。
- ❑ 连接线类：主要负责棋子中连接线的绘画。
- ❑ 英雄榜对话框类：主要负责游戏等级记录的更新。
- ❑ 背景音乐播放类：主要负责游戏中背景音乐的播放。
- ❑ 帮助对话框类：主要负责帮助提示的显示及其他辅助信息。

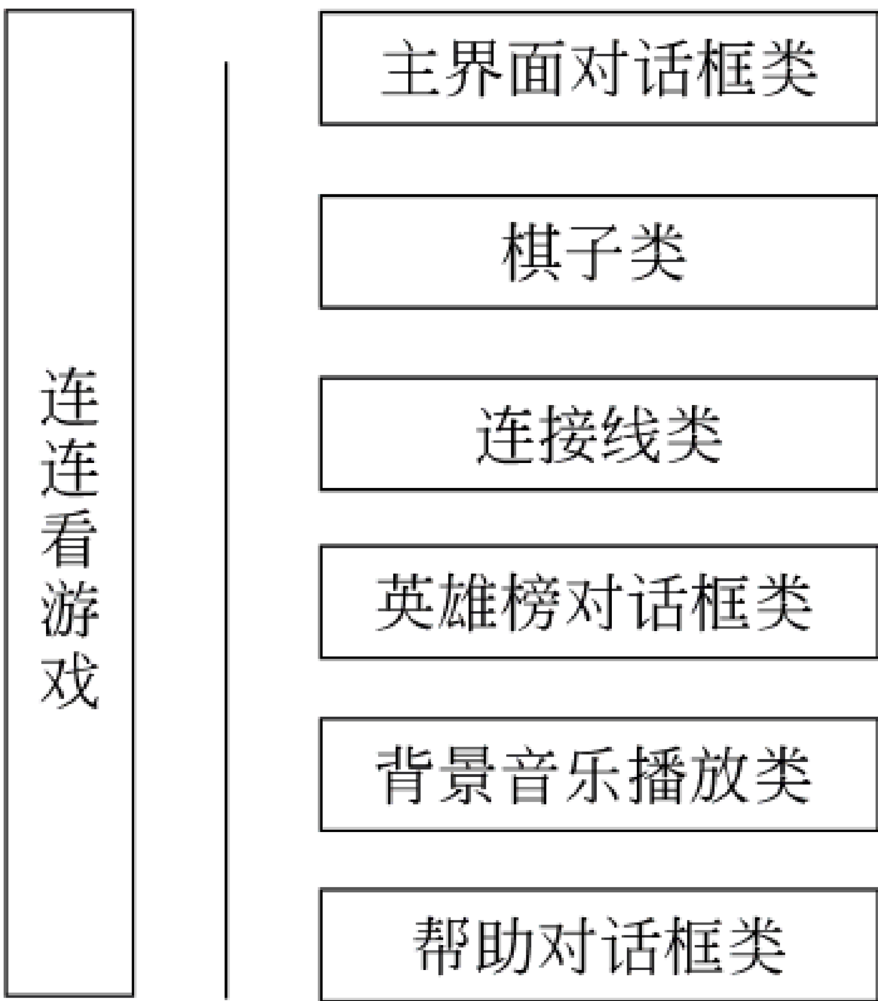


图 13.2　游戏主要类结构

13.3　连连看游戏操作界面及测试用例设计

本章将继续介绍《游戏操作界面设计文档》和《游戏测试用例文档》的编写。

13.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的连连看游戏的操作界面设计文档内容如下所述。

技巧：操作界面设计要做到让人一目了然，看文档就知道如何操作才是主要内容。

1．引言

为了让所有的项目开发人员明确连连看游戏的操作界面是如何设计的，特制定本文档用于描述本公司连连看游戏项目的游戏操作界面。

2．文档范围

包含本公司连连看游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《连连看游戏需求分析说明书》。

《连连看游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面的设计

连连看的游戏主界面设计，如图 13.3 所示。

5.2　游戏菜单结构的设计

连连看的游戏菜单设计如图 13.4 所示。

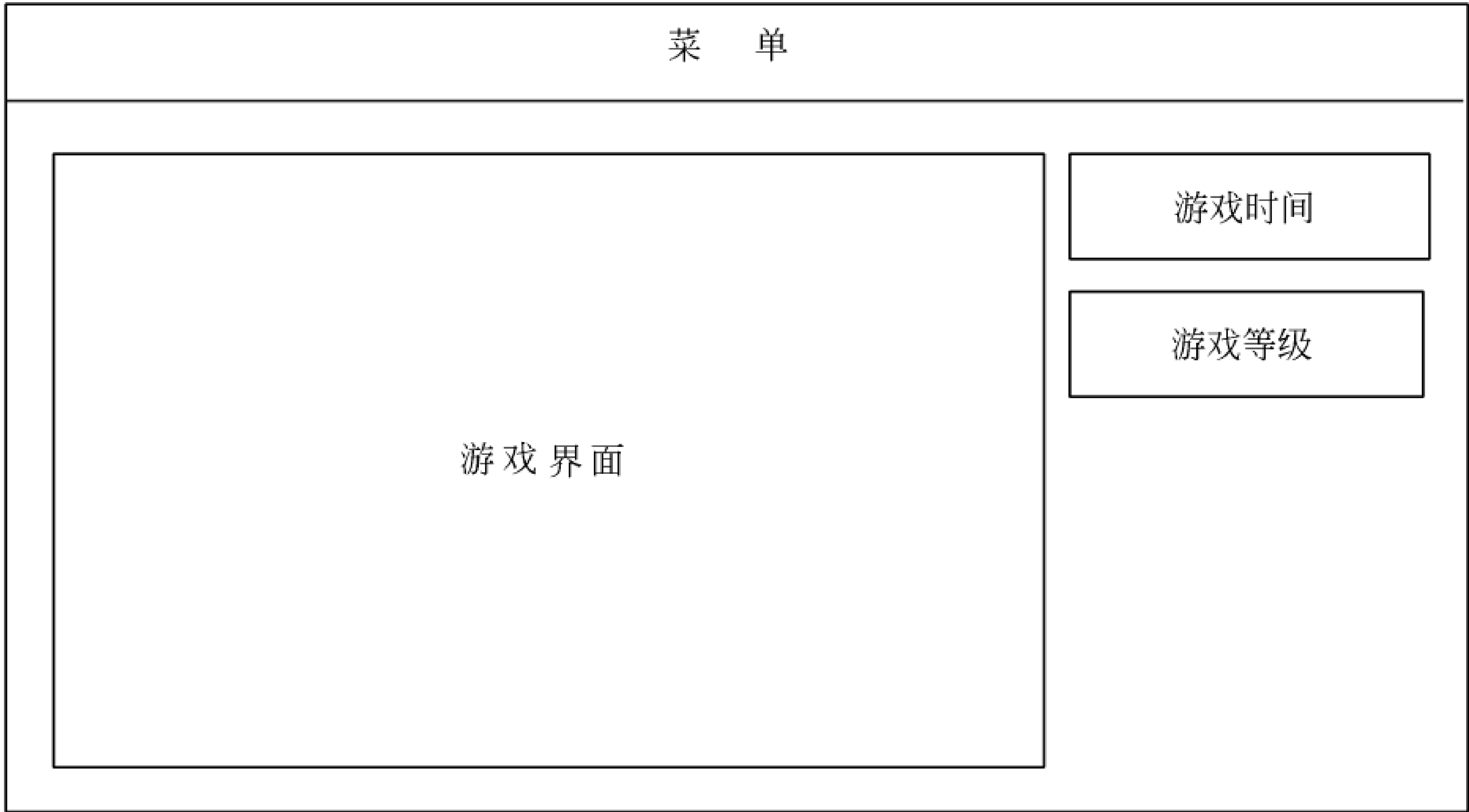


图 13.3　设计的游戏主界面

5.3　英雄榜对话框的设计

连连看英雄榜对话框的设计如图 13.5 所示。

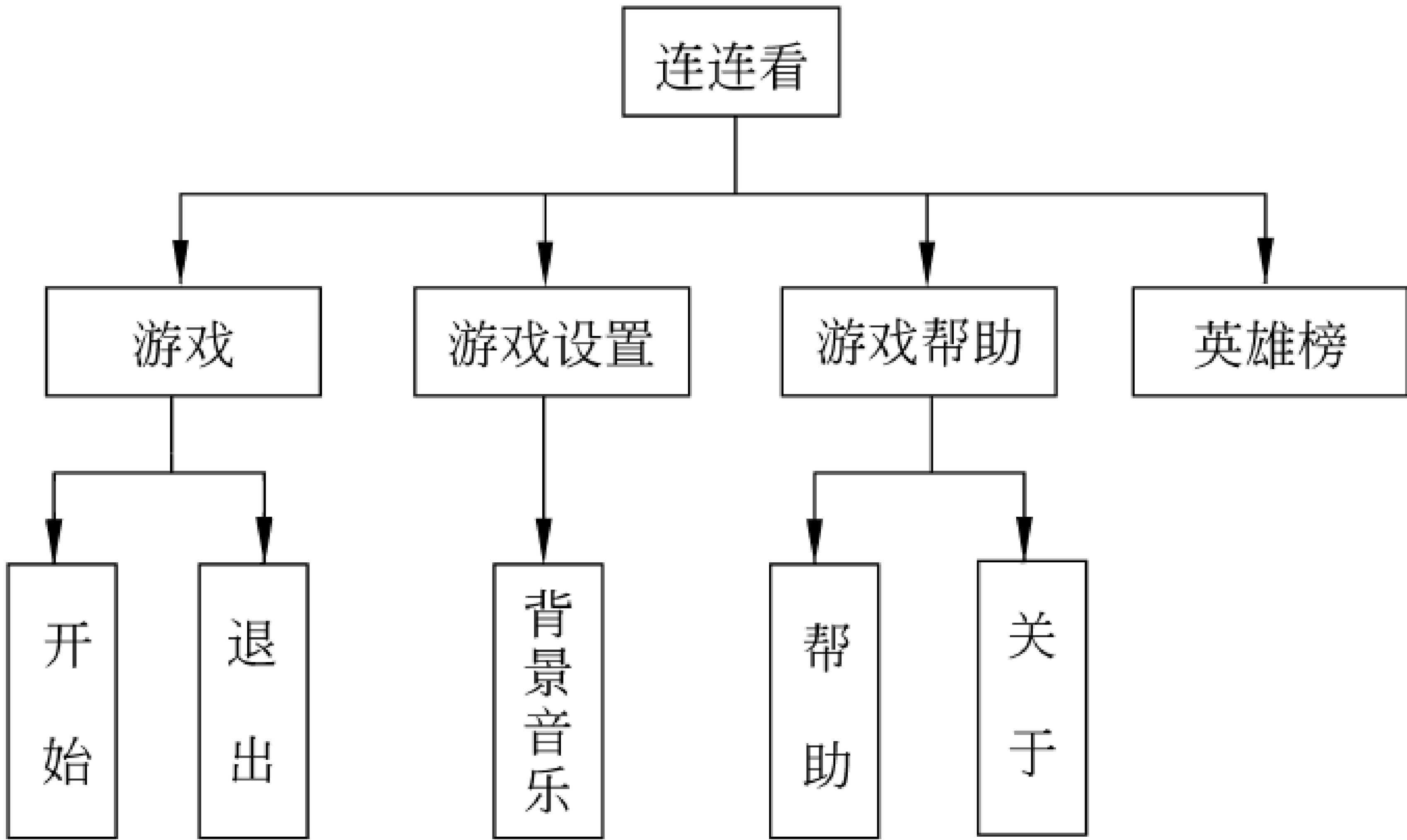


图 13.4　设计的游戏菜单结构



图 13.5　设计的“英雄榜”对话框

13.3.2　测试用例文档

测试用例文档主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的连连看游戏的测试用例文档内容如下所述。

技巧： 测试用例文档的编写一定不能由开发人员兼职编写，这样能有效提高测试用例的效率。

1．引言

本文档主要用于某公司在开展连连看游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过。方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含

6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的连连看游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行连连看游戏项目的测试和验收。

3．使用对象

本测试用例文档使用对象主要是与某公司连连看游戏开发相关的需求分析、测试和维护等部门（单位）的人员。

4．参考文献

《连连看游戏的需求分析说明书》；

《连连看游戏的概要设计文档》；

《连连看游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试环境

测试环境要求可以分为测试方法和测试工具两种。

6.1　测试方法

- 人工操作方式。

6.2　测试工具

- 鼠标、键盘。

7．测试项目

测试项目主要针对连连看中的各种功能进行整合性测试，共包含如下几个项目。

（1）主菜单和界面显示功能的测试，主要内容如表 13.2 所示。

表 13.2　主菜单和界面显示功能的测试

测试编号：1.7.1	类别：A
项　　目：连连看测试	
分 项 目：主菜单和界面显示功能的测试	
测试目的：测试连连看游戏中的菜单和界面是否正确显示	
测试配置：	
预置条件： 连连看游戏源程序已经编译完成，并可以运行； 键盘和鼠标已准备好	
测试步骤： 运行连连看程序，查看菜单和界面	
预期结果： 游戏主界面及菜单与操作设计文档中的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏主界面和菜单是否正确显示（是/否）	
测试结果： 通过/不通过	

（2）消除相同棋子功能的测试，主要内容如表 13.3 所示。

表 13.3　消除相同棋子功能的测试

测试编号：1.7.2	类别：A
项　　目：连连看测试	
分 项 目：消除相同棋子功能的测试	
测试目的：测试选中游戏中相同棋子并可以连接时，是否能被消除	
测试配置：	
预置条件： 连连看游戏已经开始	
测试步骤： 选中一对能够使用直线连接的相同棋子	
预期结果： 选中的一对棋子被消除	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 选中游戏中相同棋子并可以连接时，是否能被消除（是/否）	
测试结果： 通过/不通过	

（3）时间限制功能的测试，主要内容如表 13.4 所示。

表 13.4　时间限制功能的测试

测试编号：1.7.3	类别：A
项　　目：连连看测试	
分 项 目：时间限制功能的测试	
测试目的：测试游戏中的时间限制是否能根据游戏状态进行增长	
测试配置：	
预置条件： 连连看游戏已经开始	
测试步骤： 记住当前限制时间 消除一对棋子后，查看时间限制是否增长	
预期结果： 时间限制增长 3s	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中时间限制是否能根据游戏状态进行增长（是/否）	
测试结果： 通过/不通过	

（4）游戏升级功能的测试，主要内容如表 13.5 所示。

表 13.5　游戏升级功能的测试

测试编号：1.7.4	类别：A
项　　目：连连看测试	
分 项 目：游戏升级功能的测试	
测试目的：测试游戏中的等级在到达升级条件后，能否升级	
测试配置：	
预置条件： 当前游戏等级是 1 级； 游戏已经开始	
测试步骤： 消除完 1 等级时的全部棋子； 查看重新开始游戏时的等级显示及时间限制	
预期结果： 游戏等级变成 2； 游戏时间限制变成 110s	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的等级是否可以设置（是/否）	
测试结果： 通过/不通过	

（5）背景音乐播放功能的测试，主要内容如表 13.6 所示。

表 13.6　背景音乐播放功能的测试

测试编号：1.7.5	类别：A
项　　目：连连看测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试连连看游戏能否支持播放背景音乐	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否支持播放背景音乐（是/否）	
测试结果： 通过/不通过	

（6）帮助功能的测试，主要内容如表 13.7 所示。

表 13.7　帮助功能的测试

测试编号：1.7.6	类别：A
项　　目：连连看测试	
分 项 目：帮助功能的测试	
测试目的：测试连连看游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选中“游戏帮助”|“帮助”菜单栏	
预期结果： 出现游戏帮助提示，说明游戏操作方法	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 连连看游戏是否有帮助提示功能（是/否）	
测试结果： 通过/不通过	

（7）英雄榜功能的测试，主要内容如表 13.8 所示。

表 13.8　英雄榜功能的测试

测试编号：1.7.7	类别：A
项　　目：连连看测试	
分 项 目：英雄榜功能的测试	
测试目的：测试连连看游戏的英雄榜记录是否正确	
测试配置：	
预置条件： 玩家在游戏中已经超过上次最高记录分数	
测试步骤： 等待游戏结束； 在弹出的对话框中输入玩家的大名； 选中“英雄榜”菜单	
预期结果： 在弹出的对话框中，查看等级及大名是否被记录	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 连连看游戏的英雄榜记录是否正确（是/否）	
测试结果： 通过/不通过	

（8）消除提示功能的测试，主要内容如表 13.9 所示。

表 13.9　消除提示功能的测试

测试编号：1.7.8	类别：A
项　　目：连连看测试	
分 项 目：消除提示功能的测试	
测试目的：测试游戏中的消除提示功能是否正确	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 按下键盘上的快捷键 F5	
预期结果： 出现可以配对消除棋子提示	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的消除提示功能是否正确（是/否）	
测试结果： 通过/不通过	

（9）棋子换盘功能的测试，主要内容如表 13.10 所示。

表 13.10　棋子换盘功能的测试

测试编号：1.7.9	类别：A
项　　目：连连看测试	
分 项 目：棋子换盘功能的测试	
测试目的：测试游戏中的棋子换盘功能是否正确	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 消除若干对棋子后，按下快捷键 F5	
预期结果： 棋盘上的各棋子的位置发生随机变化	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的棋子换盘功能是否正确（是/否）	
测试结果： 通过/不通过	

13.4　连连看游戏的详细设计

本节主要对连连看游戏中的功能进行详细讲解。为了突出重点，在这里不描述详细设计文档的格式。

技巧： 以流程图或者算法描述语言来表达设计思想，比只有流程图更高效。

13.4.1　游戏各功能的设计描述

在连连看游戏中，大致可以分为 8 个功能模块，如下所示。

1．时间限制模块的算法设计

时间限制模块的算法主要分为如下几个步骤：

（1）在游戏开始时，设置当前限制时间（nOverTime）为 60s。

（2）设定一个时间定时器 TIMER1，时间间隔为 1000ms。

（3）当每一次时间间隔到时，就把当前限制时间减少 1s。

（4）如果游戏中有一对棋子消除时，就把限制时间增加 3。例如，当前限制时间为 50s。这时有棋子被消除，那么限制时间变成 53s。

（5）如果限制时间变成 0s，说明游戏结束，弹出结束提示对话框。如果当前游戏等级超过记录等级，还要弹出英雄榜对话框对话。

2．消除相同棋子模块的算法设计

消除相同棋子模块的算法主要分为如下几个步骤：

（1）当鼠标单击棋子时，保存到棋子坐标及类型到“第一次选中变量”中。

（2）得到鼠标第二次选中的棋子坐标及类型。

（3）比较两次棋子类型，如果相同，转步骤（4）。如果不相同，转步骤（5）。

（4）画一条连接线，并消除这对棋子，退出等待下一次鼠标选择。

（5）把鼠标第二次选中的棋子坐标及类型赋值给“第一次选中变量”。

3．游戏升级模块的算法设计

游戏升级模块的算法主要分为如下几个步骤：

（1）保存当前游戏等级和限制时间初始值。

（2）当游戏中的棋子全部被消除完毕时，就把当前游戏等级增加一级。

（3）把限制时间的初始值减少 5s。

（4）调用重新开始游戏接口函数。

4．消除提示模块的算法设计

消除提示模块的算法主要分为如下几个步骤：

（1）判断提示次数变量的值，如果等于 0，结束。

（2）查找当前棋盘中相同的棋子。

（3）在相同棋子中，查找可以消除的棋子。

（4）给出提示连线，提示次数变量减少 1。

5．棋子换盘模块的算法设计

棋子换盘模块的算法主要分为如下几个步骤：

（1）保存当前棋盘数组中的数据到临时数组中。

（2）循环地从临时数组中，随机取出数据保存到棋盘数组中。

6．英雄榜模块的算法设计

英雄榜模块的算法主要分为如下几个步骤：

（1）读取配置文件（setup.ini），得到并显示当前最高等级及大名。

（2）在用户结束游戏时，比较当前用户等级和最高等级。如果高于配置文件中的最高等级，就弹出“英雄榜”对话框，要求输入大名，并连同用户的等级保存到配置文件中。

7．音乐播放模块的算法设计

音乐播放模块比较简单，只需要在用户选择音乐播放时，把音乐资源载入程序并播放。

8．帮助类模块的算法设计

帮助类模块的算法也比较简单，只要把相应的对话框资源显示出来即可。

13.4.2　游戏各功能流程图

在这里只给出消除及限制时间功能的详细流程图，如图 13.6 所示。

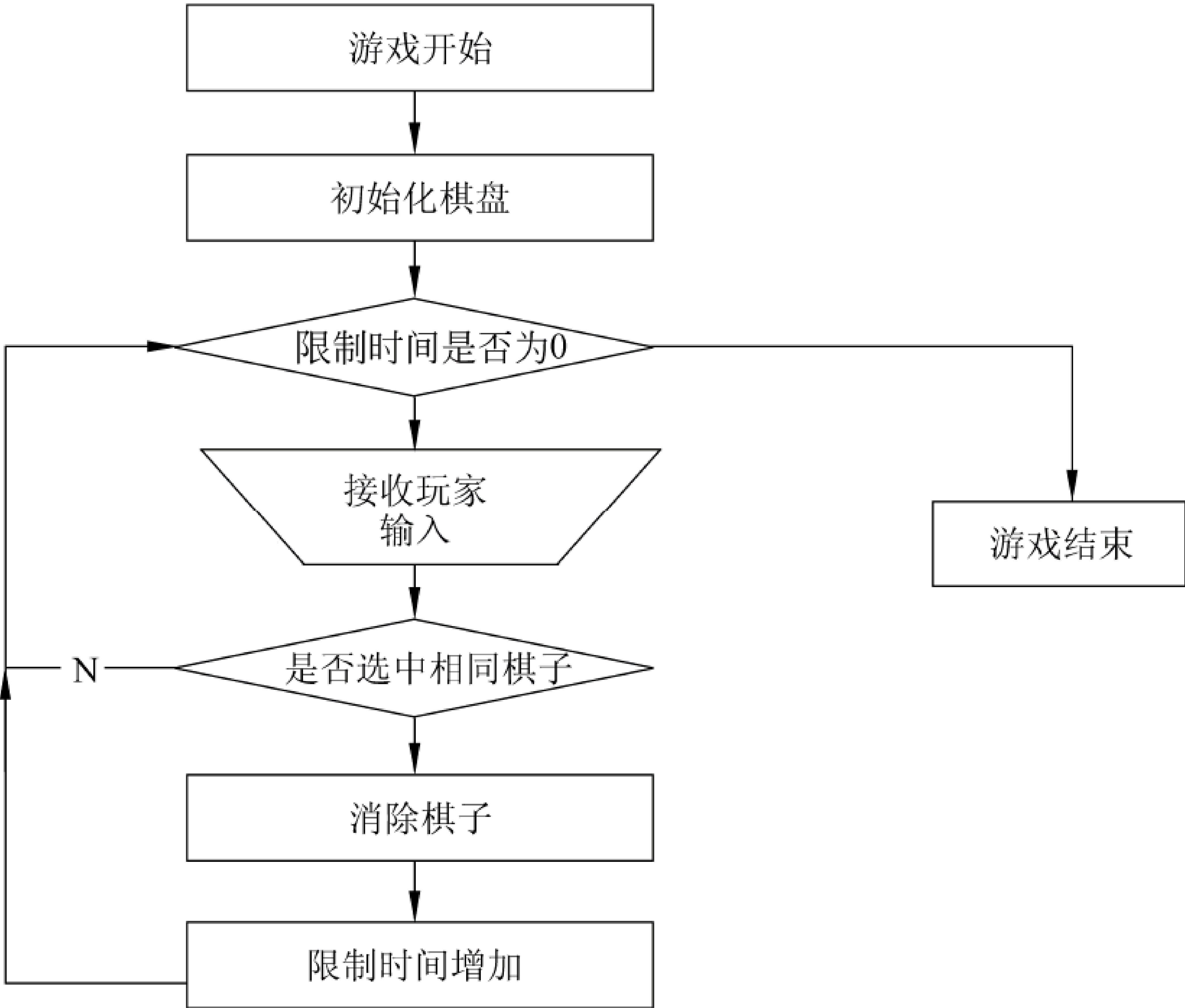


图 13.6　消除及限制时间功能流程图

13.5 连连看游戏的界面实现

连连看游戏的 Visual C++工程采用 MFC 对话框模式进行开发。本节主要讲解连连看游戏各个功能模块的代码实现。

13.5.1 游戏菜单的实现

在连连看游戏中，通过如下几个步骤即可实现游戏的菜单。

（1）在连连看游戏工程的资源中添加一个菜单资源，其属性如表 13.11 所示。

表 13.11 主菜单属性

ID	类 别	说 明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
IDR_EXIT_GAME	菜单栏	退出游戏
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于
IDR_HERO_LIST	菜单栏	英雄榜

（2）给每个菜单栏添加响应函数到 CLlkDlg 类中。

（3）菜单响应函数的实现，如代码 13.1 所示。

代码 13.1 菜单响应函数的实现

```
01  // llkDlg.cpp CllkDlg类实现源文件
02
03
04  #include "stdafx.h"                 //插入头文件
05  #include "llk.h"
06  #include "llkDlg.h"                 //插入类声明头文件
07
08  #include "HeroDlg.h"                //插入英雄榜对话框类头文件
09  #include "HelpDlg.h"                //插入帮助对话框类头文件
10  ...//省略部分代码，请查阅光盘上的源代码
11  BEGIN_MESSAGE_MAP(CLlkDlg, CDialog)  //成员函数映射
12      ON_WM_SYSCOMMAND()
13      ON_WM_PAINT()
14      ON_WM_QUERYDRAGICON()
15      ON_COMMAND(IDR_ABOUT, OnAbout)  //“关于”菜单栏响应函数
16                                      //“退出”菜单栏响应函数
17      ON_COMMAND(IDR_EXIT_GAME, OnExitGame)
18      ON_COMMAND(IDR_HELP, OnHelp)    //“帮助”菜单栏响应函数
19                                      //“英雄榜”菜单栏响应函数
20      ON_COMMAND(IDR_HERO_LIST, OnHeroList)
21                                      //“播放音乐”菜单栏响应函数
```

```
22      ON_COMMAND(IDR_PLAY_MUSIC, OnPlayMusic)
23                                    //“开始”菜单栏响应函数
24      ON_COMMAND(IDR_START_GAME, OnStartGame)
25  END_MESSAGE_MAP()
26
27  BOOL CLlkDlg::OnInitDialog()      //初始化对话框
28  {
29      CDialog::OnInitDialog();      //调用基类函数初始化
30                                    //得到系统菜单
31      CMenu* pSysMenu = GetSystemMenu(FALSE);
32      if (pSysMenu != NULL)         //判断加载是否成功
33      {
34          CString strAboutMenu;
35          strAboutMenu.LoadString(IDS_ABOUTBOX);
36          if (!strAboutMenu.IsEmpty())
37          {
38              pSysMenu->AppendMenu(MF_SEPARATOR);
39              pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
40          }
41      }
42
43      SetIcon(m_hIcon, TRUE);       //设置大图标
44      SetIcon(m_hIcon, FALSE);      //设置小图标
45                                    //加载菜单
46      m_main_menu.LoadMenu(IDR_MAIN_MENU);
47      SetMenu(&m_main_menu);        //设置主菜单
48
49      m_bStart=FALSE;               //设置游戏开始标志变量值
50
51      return TRUE;                  //返回真，初始化成功
52  }
53
54  void CLlkDlg::OnAbout()           //“关于”菜单栏响应函数实现
55  {
56      CAboutDlg dlg;                //创建关于对话框对象
57      dlg.DoModal();                //弹出关于对话框
58  }
59
60  void CLlkDlg::OnExitGame()        //“退出”菜单栏响应函数实现
61  {
62      CDialog::OnCancel();          //调用基类退出函数
63  }
64
65  void CLlkDlg::OnHelp()            //“帮助”菜单栏响应函数实现
66  {
67      CHelpDlg dlg;                 //创建帮助对话框类对象
68      dlg.DoModal();                //弹出帮助对话框
69  }
70
71  void CLlkDlg::OnHeroList()        //“英雄榜”菜单栏响应函数实现
72  {
73      CHeroDlg dlg;                 //创建英雄榜对话框类对象
74      dlg.DoModal();                //弹出英雄榜对话框
75  }
76
77  void CLlkDlg::OnPlayMusic()       //“播放音乐”菜单栏响应函数实现
78  {
```

```
79      //判断播放音乐菜单当前状态
80      BOOL bCheck = (BOOL)m_main_menu.GetMenuState(IDR_PLAY_MUSIC,
81                              MF_CHECKED);
82
83      if(m_bStart)
84      {                               //游戏如果开始才允许播放音乐
85          if(bCheck)
86          {
87              m_main_menu.CheckMenuItem(IDR_PLAY_MUSIC,
88                  MF_BYCOMMAND | MF_UNCHECKED);
89          }
90          else
91          {
92              m_main_menu.CheckMenuItem(IDR_PLAY_MUSIC,
93                  MF_BYCOMMAND | MF_CHECKED);
94          }
95
96          PlayBackMusic(!bCheck);     //调用播放背景音乐功能函数
97      }
98  }
99
100 void CLlkDlg::OnStartGame()         //“开始”菜单栏响应函数实现
101 {
102     m_llk_game.StartGame();         //调用主游戏类开始游戏成员函数接口
103 }
```

代码解析：本代码主要实现菜单中的菜单项选择的响应。其中对话框初始化函数是对游戏开始标志进行初始化设置。代码第 102 行，是调用 m_llk_game 对象的游戏开始函数开始游戏。

13.5.2　游戏帮助对话框的实现

连连看游戏中的帮助是使用一个对话框来实现的，实现的步骤如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 13.7 所示。

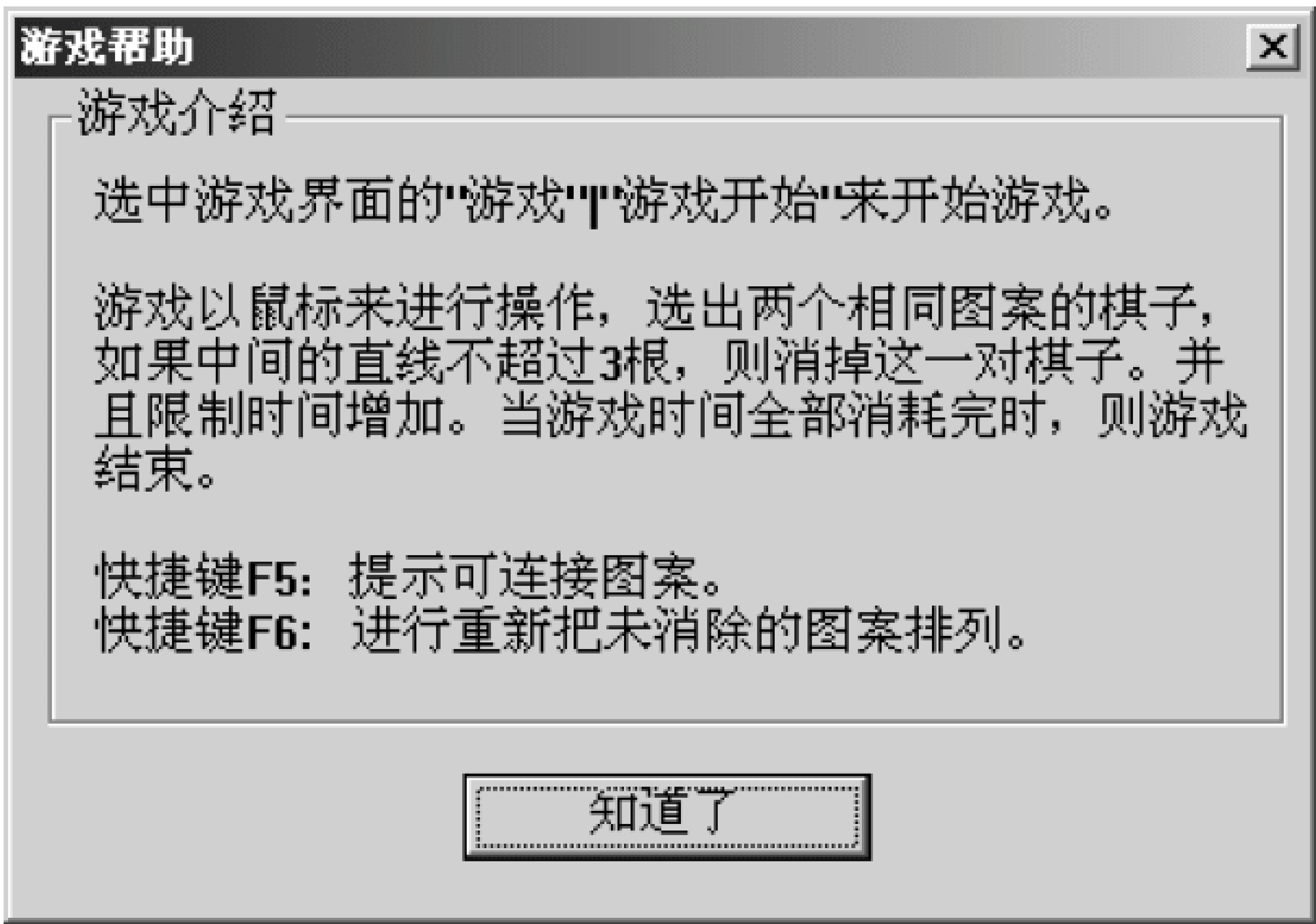


图 13.7　帮助对话框

（2）编写一个 CHelpDlg 对话框类，主要是加载 IDD_HELP 对话框资源。通过资源中的文字说明对游戏操作方法进行描述。同时只包含单击“知道了”按钮的响应函数。其类声明如代码 13.2 所示。

代码 13.2　CHelpDlg 对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
04  // HelpDlg.h CHelpDlg 类声明头文件
05
06
07  /////////////////////////////////////////////
08  // CHelpDlg 对话框类
09
10  class CHelpDlg : public CDialog            //公共继承于 CDialog 类
11  {
12  public:
13      CHelpDlg(CWnd* pParent = NULL);        //构造函数
14
15  //对话框资源
16      enum { IDD = IDD_HELP };               //加载资源
17
18  //重载函数
19      protected:
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                   //单击“确定”按钮响应函数声明
25      DECLARE_MESSAGE_MAP()
26  };
27
28  #endif
```

（3）CHelpDlg 对话框类中，需要实现对话框类的构造函数、析构函数和“知道了”按钮响应函数。其代码如代码 13.3 所示。

代码 13.3　CHelpDlg 对话框类的实现

```
01  // HelpDlg.cpp CHelpDlg 类的实现源文件
02
03
04  #include "stdafx.h"                        //插入头文件
05  #include "llk.h"
06  #include "HelpDlg.h"                       //插入类声明头文件
07
08  /////////////////////////////////////////////
09  // CHelpDlg 对话框类实现
10
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/)//构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);
19  }
20
21  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)
22  END_MESSAGE_MAP()
23
```

```
24  /////////////////////////////////////////////
25  // CHelpDlg 消息响应函数
26
27  void CHelpDlg::OnOK()                          //单击“知道了”按钮响应函数
28  {
29      CDialog::OnOK();
30  }
```

代码解析：主要是 CHelpDlg 对话框类的实现代码，其中 OnOK()函数是对界面中“知道了”按钮的响应函数的实现。

13.5.3　游戏英雄榜对话框的实现

连连看游戏英雄榜的实现，分为如下几个步骤：

（1）创建一个对话框资源，并添加相应的控件，如图 13.8 所示。

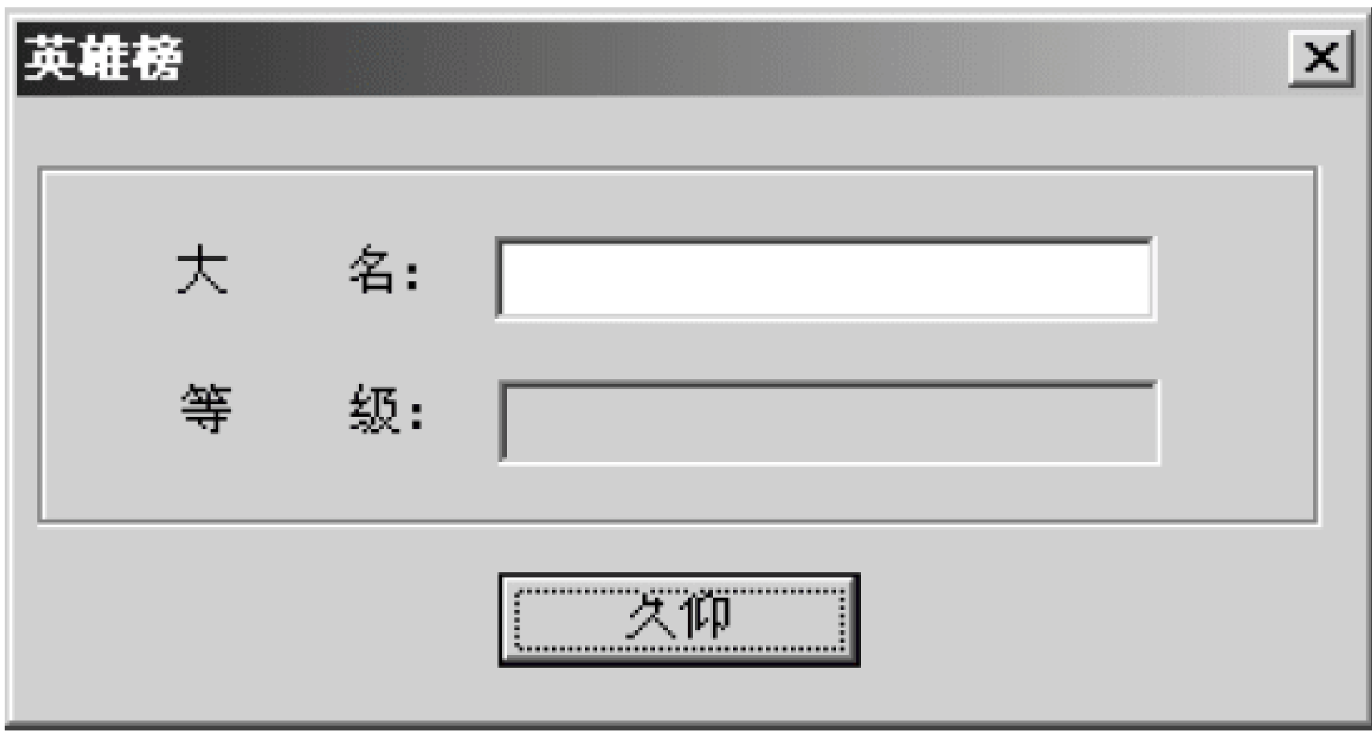


图 13.8　英雄榜对话框资源

（2）配置（setup.ini）文件格式如下：

```
[HERO]
name=XXX
level=0
```

（3）添加 CHeroDlg 类，其中需要包含 IDD_HERO_DLG 对话框资源和“设置可写记录标志”接口函数声明，用于外部函数调用时，设置是否对配置文件进行写操作。类的声明如代码 13.4 所示。

代码 13.4　CHeroDlg 对话框类的实现

```
01  #if !defined(AFX_HERODLG_H__)
02  #define AFX_HERODLG_H__
03
04  // HeroDlg.h 头文件
05
06  /////////////////////////////////////////////
07  // CHeroDlg 对话框类声明
08
09  class CHeroDlg : public CDialog
10  {
11  public:
12      void SetWriteFlg(BOOL bflg);            //接口函数，设置可记录标志变量
13      CHeroDlg(CWnd* pParent = NULL);         //构造函数
14
15      enum { IDD = IDD_HERO_LIST };           //对话框资源
16      int     m_level;                        //保存等级变量
```

```
17      CString m_name;                           //保存姓名变量
18
19      public:
20      virtual int DoModal();                    //弹出对话框函数声明
21      protected:
22      virtual void DoDataExchange(CDataExchange* pDX);
23
24  protected:
25
26      virtual void OnOK();                      //单击“久仰”按钮响应函数声明
27
28      DECLARE_MESSAGE_MAP()
29  private:
30      BOOL m_bWriteflg;                         //记录标志变量
31  };
32
33  #endif
```

（4）CHeroDlg 类的实现中通过调用系统 API 函数，来对配置文件进行读写操作。而“设置读写标志”接口函数，是对类的一个成员变量 m_bWrite 进行赋值操作，达到写入或者读取的区分。其代码如代码 13.5 所示。

代码 13.5　CHeroDlg 类的实现

```
01  // HeroDlg.cpp 源文件
02  #include "stdafx.h"                       //插入头文件
03  #include "llk.h"
04  #include "HeroDlg.h"                      //插入类声明头文件
05
06  ///////////////////////////////////////////
07  // CHeroDlg 对话框
08
09  CHeroDlg::CHeroDlg(CWnd* pParent /*=NULL*/)//构造函数
10      : CDialog(CHeroDlg::IDD, pParent)
11  {
12      m_bWriteflg = FALSE;                  //初始化写标志变量为假
13  }
14
15  void CHeroDlg::DoDataExchange(CDataExchange* pDX)
16  {
17      CDialog::DoDataExchange(pDX);         //变量与资源映射
18      //{{AFX_DATA_MAP(CHeroDlg)
19      DDX_Text(pDX, IDC_LEVEL_EDIT, m_level);
20      DDX_Text(pDX, IDC_NAME_EDIT, m_name);
21      //}}AFX_DATA_MAP
22  }
23
24  BEGIN_MESSAGE_MAP(CHeroDlg, CDialog)
25      ON_BN_CLICKED(IDOK_BTN, OnBtn)        //按钮与函数映射
26  END_MESSAGE_MAP()
27
28  ////////////////////////////////////////////
29  // CHeroDlg 消息句柄
30
31  void CHeroDlg::SetWriteFlg(BOOL bflg)     //设置写入标志
32  {
33      m_bWriteflg = bflg;
34  }
```

```
35
36  int CHeroDlg::DoModal()                          //弹出对话框
37  {
38      char pszTmp[128] = {0};
39
40      //读取配置文件
41      GetPrivateProfileString("HERO", "name", "0",
42          pszTmp, 127, ".\\hero.ini");             //读入姓名
43      m_name = CString(pszTmp);
44
45      if(!m_bWriteflg)
46      {
47          GetPrivateProfileString("HERO", "level", "0",
48              pszTmp, 127, ".\\hero.ini");         //读入等级
49          m_level = atoi(pszTmp);
50      }
51
52      return CDialog::DoModal();
53  }
54
55  void CHeroDlg::OnBtn()                           //按钮响应
56  {
57      UpdateData(TRUE);
58      if(m_bWriteflg)
59      {
60          CString tmp;
61                                                   //写入姓名
62          WritePrivateProfileString("HERO", "name", m_name, ".\\hero.ini");
63          tmp.Format("%d", m_level);
64                                                   //写入等级
65          WritePrivateProfileString("HERO", "level", tmp, ".\\hero.ini");
66      }
67      m_bWriteflg = FALSE;
68
69      CDialog::OnOK();
70  }
71
72  BOOL CHeroDlg::OnInitDialog()                    //初始化对话框
73  {
74      CDialog::OnInitDialog();
75
76      if(m_bWriteflg)
77      {                                            //当为写入时，改变按钮名称
78          SetDlgItemText(IDOK_BTN, "记录");
79      }
80
81      return TRUE;
82  }
```

代码解析：第 41 行和第 42 行是调用系统 API 函数 GetPrivateProfileString()读取文件中指定名称的参数值。代码第 62 行是调用系统 API 函数 WritePrivateProfileString()写入文件中指定的名称的参数值。

13.5.4　游戏播放背景音乐的实现

游戏背景音乐播放，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩

家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。要实现这个功能，需要如下几个步骤。

（1）在工程文件中，添加“winmm.lib”静态库文件及头文件，参见第 5.4 节。

（2）实现 CLlkDlg 类中的 PlayBackMusic()成员函数，其代码如代码 13.6 所示。

代码 13.6　CLlkDlg 类的 PlayBackMusic 成员函数实现

```
01  #include <mmsystem.h>                          //插入系统 API 头文件
02  ...
03  void CLlkDlg::PlayBackMusic(BOOL bCheck)
04  {
05      //指定文件并播放
06      if(bCheck)
07      {                                          //播放指定音乐文件
08          sndPlaySound("music.wav",SND_ASYNC);
09      }
10      else
11      {                                          //停止播放
12          sndPlaySound(NULL,SND_PURGE);
13      }
14  }
```

代码解析：第 8 行和第 12 行是调用 sndPlaySound()系统函数来实现播放和停止音乐功能，其中第一个参数为指定播放的文件名称，第二个参数为控制参数。

13.6　连连看游戏的核心算法设计与实现

在前面的章节里已经讲解了连连看游戏的菜单和各种对话框的实现。本节将对连连看游戏的核心算法的设计和实现进行讲解。

13.6.1　主对话框类的设计

连连看的主对话框类，主要负责显示游戏界面、等级、时间显示及快捷键调用等内容。其主要有如下几个处理模块。

1．主菜单处理模块

主菜单处理模块比较简单，只需要创建一个菜单对象，并在对话框初始化函数中进行加载相应资源并设置到对话框中。

2．连接提示处理模块

连接提示模块，主要调用棋子类中的查找有效连接接口函数，通过返回的两个图标相同的棋子位置来设置两个棋子的状态为按下。

3．换盘处理模块

换盘处理模块，主要是把棋盘数组中的数据重新进行随机排列。

4．初始化棋盘数据模块

初始化棋盘数据模块，利用随机数，随机地选择在棋盘数组中插入棋子数据。每一类棋子生成 2 组 4 个相同的数据。

5．游戏信息处理模块

游戏信息处理模块，主要利用设置定时器，每一秒进行更新显示区域。在这些区域中，以绘图方式进行输出。

13.6.2　主对话框类的实现

实现主对话框类 CLlkDlg 需要如下几个步骤。

（1）声明主对话框 CLlkDlg 类，其中包含连接提示、显示棋盘、显示数据、换盘、初始化游戏及棋盘数据函数。其代码如代码 13.7 所示。

代码 13.7　主对话框类 CLlkDlg 的声明

```
01  // llkDlg.h  CLikDlg 类声明的头文件
02
03  #if !defined(AFX_LLKDLG_H__)
04  #define AFX_LLKDLG_H__
05
06  ///////////////////////////////////////////
07  // CLlkDlg 对话框类
08
09  #include "LineStatic.h"                      //插入连接线类的头文件
10  #include "ChessMan.h"                        //插入棋子类的头文件
11
12  class CLlkDlg : public CDialog               //定义主对话框类
13  {
14  public:
15      void RefreshMap();                       //更新棋盘数组
16      void Exchange(int map[MAXX][MAXY]);      //换盘函数
17      void CallHint();                         //调用提示接口函数
18      void CallExchange();                     //调用换盘接口函数
19      void ShowMap(int map[MAXX][MAXY]);       //显示棋盘数据
20      void InitMap(int map[MAXX][MAXY]);       //初始化棋盘数据
21      void ShowMsg(CRgn * rgn);                //显示各种数据
22      void Start(int nlevel);                  //开始游戏
23      void PlayBackMusic(BOOL bflag);          //播放背景音乐
24      void isHighLevel();                      //超过记录判断
25      CLlkDlg(CWnd* pParent = NULL);           //构造函数
26
27      CChessMan * m_p;                         //棋子指针
28      int map[MAXX][MAXY];                     //棋盘数组
29      BOOL    m_bStart;                        //游戏开始状态
30      int     m_nLevel;                        //当前等级限制
31      int     m_timePoint;                     //时间限制
32      CRgn    m_MsgRgn;                        //信息区范围
33      CPtrArray m_cmGroup;                     //棋子数组
34      int     m_hintNum;                       //提示次数限制
```

```
35      int     m_exchangeNum;                  //换盘次数限制
36      int     m_nHighLevel;                   //最高等级
37
38      enum { IDD = IDD_LLK_DIALOG };          //资源映射
39      CStatic m_method;
40      CLineStatic m_line;
41      CStatic m_bkPic;
42
43      public:                                 //消息截获函数
44      virtual BOOL PreTranslateMessage(MSG* pMsg);
45      protected:
46      virtual void DoDataExchange(CDataExchange* pDX);    //资源加载函数
47      //}}AFX_VIRTUAL
48
49  protected:
50      HICON m_hIcon;                          //图标句柄
51
52      CMenu m_main_menu;                      //主菜单对象
53
54      int m_hintP2;                           //第 2 个转折点
55      int m_hintP1;                           //第 1 个转折点
56
57      virtual BOOL OnInitDialog();            //对话框初始化函数
58      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
59      afx_msg void OnPaint();
60      afx_msg HCURSOR OnQueryDragIcon();
61      afx_msg void OnAbout();
62      afx_msg void OnExitGame();
63      afx_msg void OnHelp();
64      afx_msg void OnHeroList();
65      afx_msg void OnPlayMusic();
66      afx_msg void OnStartGame();
67      afx_msg void OnTimer(UINT nIDEvent);
68      //}}AFX_MSG
69      DECLARE_MESSAGE_MAP()
70  };
71
72  #endif
```

（2）添加主对话框类 CLlkDlg 的实现，其中包含基本的初始化对话框函数、游戏开始处理函数及背景音乐播放功能函数。其代码如代码 13.8 所示。

代码 13.8　主对话框类 CLlkDlg 的实现

```
01  // llkDlg.cpp CLlkDlg 类的实现
02
03  #include "stdafx.h"                     //插入头文件
04  #include "llk.h"
05  #include "llkDlg.h"                     //插入类声明头文件
06
07  #include "HeroDlg.h"                    //插入英雄榜对话框类头文件
08  #include "HelpDlg.h"                    //插入帮助对话框类头文件
09
10  #include <mmsystem.h>                   //插入多媒体 API 声明头文件
11
12  ...                                     //省略部分重复代码，参见代码 13.1
13  BOOL CLlkDlg::OnInitDialog()            //对话框类初始化
14  {
```

```
15      CDialog::OnInitDialog();                  //调用基类初始化函数
16
17      SetIcon(m_hIcon, TRUE);                   //设置大图标
18      SetIcon(m_hIcon, FALSE);                  //设置小图标
19                                                //加载主菜单
20      m_main_menu.LoadMenu(IDR_MAIN_MENU);
21
22      SetMenu(&m_main_menu);                    //设置主菜单
23
24      m_nLevel = 1;                             //初始化游戏等级变量
25                                                //重新设置主对话框大小及位置
26      SetWindowPos(NULL, 100, 100, 800, 600, SWP_SHOWWINDOW|SWP_NOZORDER);
27                                                //重新设置进程提示标签的大小及位置
28      m_method.SetWindowPos(NULL, 300, 500, 200, 25,
29                      SWP_SHOWWINDOW|SWP_NOZORDER);
30      m_bkPic.SetWindowPos(NULL, 130, 70, 480, 280,
31                      SWP_SHOWWINDOW|SWP_NOZORDER);
32      m_line.SetWindowPos(NULL, 90, 30, 560, 360,
33                      SWP_SHOWWINDOW|SWP_NOZORDER);
34                                                //设置消息输出区的范围
35      m_MsgRgn.CreateRectRgn(150, 20, 550, 40);
36
37      m_bStart = false;                         //设置游戏开始变量为假
38      return TRUE;                              //初始化成功
39  }
40  ...                                           //省略部分重复代码
41  void CLlkDlg::OnStartGame()                   //菜单“开始游戏”栏响应函数
42  {
43      char pszTmp[128] = {0};
44
45      m_hintNum = 3;                            //设置提示限制次数
46      m_exchangeNum = 3;                        //设置换盘限制次数
47      m_line.m_lineNum = 0;                     //设置连接线数
48                                                //提示开始游戏
49      m_method.SetWindowText("开 始 游 戏!");
50
51      Start(1);                                 //调用开始游戏接口函数
52                                                //读最高游戏记录中等级
53      GetPrivateProfileString("HERO", "level", "1",
54              pszTmp, 127, ".\\setup.ini");
55      m_nHighLevel = atoi(pszTmp);              //字符转换成数字
56  }
57  ////////////////////////////////////////
58  //播放背景音乐
59  ////////////////////////////////////////
60  void CLlkDlg::PlayBackMusic(BOOL bflag)
61  {
62      //指定文件并播放
63      if(bflag)
64      {                                         //播放音乐
65          sndPlaySound("music.wav",SND_ASYNC);
66      }
67      else
68      {                                         //停止播放
69          sndPlaySound(NULL,SND_PURGE);
70      }
71  }
```

```
72  /////////////////////////////////////
73  //游戏开始
74  /////////////////////////////////////
75  void CLlkDlg::Start(int level)
76  {
77      CString str;
78
79      m_nLevel = level;
80
81      m_timePoint = 130 - level * 10;     //设置每个等级的限制时间
82
83      str.Format("res\\b%d.bmp", m_nLevel); //根据等级设置背景图片
84      m_bkPic.ModifyStyle(0,SS_BITMAP|SS_CENTERIMAGE);
85      HBITMAP m_bkBmp = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
86      str, IMAGE_BITMAP, 0, 0, LR_CREATEDIBSECTION|LR_LOADFROMFILE);
87      m_bkPic.SetBitmap(m_bkBmp);         //设置图片
88      InitMap(map);                       //初始化棋盘数据
89      ShowMap(map);                       //显示棋盘中的棋子
90      m_p = NULL;                         //初始化棋子指针
91      CRgn rgn;
92      rgn.CreateRectRgn(420, 0, 560, 160);
93      m_MsgRgn.CopyRgn(&rgn);             //设置消息显示区域范围
94      ShowMsg(&m_MsgRgn);                 //更新消息显示区域
95      KillTimer(1);
96      SetTimer(1, 1000, NULL);            //设置计时器
97      m_bStart = true;
98  }
```

代码解析：第 75～98 行是游戏开始函数的实现过程，其中第 83 行是根据当前等级设置背景图片路径，然后在第 85 行创建相应的位图句柄，最后在第 87 行将该句柄设置到背景对象中。代码第 88 行是调用初始化函数初始化棋盘数据。第 89 行调用显示棋盘函数实现棋子与背景的显示。

（3）给主对话框类添加显示函数、分数和游戏等级信息更新函数及接收玩家键盘输入处理函数。其代码如代码 13.9 所示。

代码 13.9　显示及输入处理函数

```
01  //信息显示
02  void CLlkDlg::ShowMsg(CRgn *rgn)
03  {
04      CClientDC *pDC = (CClientDC *)GetDC();
05      CFont font;
06      CString str;
07      //设置字体
08      if(0==font.CreatePointFont(150,"Comic Sans MS"))
09      {
10          AfxMessageBox("Can't Create Font");
11      }
12      pDC->SelectObject(&font);
13      pDC->SetTextColor(RGB(39,244,10)); //设置字体颜色及其背景颜色
14      pDC->SetBkColor(RGB(0,0,0));
15
16      str.Format("游戏等级: %d",m_nLevel);
17      pDC->TextOut(650,420,str);          //输出游戏等级
18
19      str.Format("剩余时间: %d", m_timePoint);
```

```
20      pDC->TextOut(650, 450, str);        //输出剩余时间
21      str.Format("提示次数: %d",m_hintNum);
22      pDC->TextOut(650,480,str);          //输出提示次数
23
24      str.Format("换盘次数: %d", m_exchangeNum);
25      pDC->TextOut(650, 510, str);        //输出换盘次数
26
27  }
28  //转换棋盘数组中的数据为棋子并显示出来
29  void CLlkDlg::ShowMap(int map[][MAXY])
30  {
31      this->Invalidate();
32      int i, j;
33      POINT p;
34      CString str;
35      for(i=0; i<m_cmGroup.GetSize(); i++)
36      {                                   //清除原有棋子
37          delete (CChessMan *)m_cmGroup.GetAt(i);
38      }
39      m_cmGroup.RemoveAll();
40      for(i=1; i<MAXX-1; i++)
41      {
42          for(j=1; j<MAXY-1; j++)
43          {
44              p.x = i;
45              p.y = j;
46              m_cmGroup.Add(new CChessMan(map[i][j], p));
47          }
48      }
49      for(i=0; i<(MAXX-2)*(MAXY-2); i++)
50      {
51          CChessMan *btn = (CChessMan *)m_cmGroup.GetAt(i);
52                                          //创建棋子按钮
53          btn->Create(str, WS_CHILD|BS_BITMAP,
54              CRect(130+(i%(MAXY-2))*40, 70+(i/(MAXY-2))*40,
55              170+(i%(MAXY-2))*40, 110+(i/(MAXY-2))*40),
56              this, IDC_BLOCK+i);
57          if(btn->m_id)                   //如果为 0 则不显示
58          {
59              str.Format("res\\%d.bmp", btn->m_id);
60          HBITMAP m_fkBmp = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
61          str, IMAGE_BITMAP, 0, 0,
62          LR_CREATEDIBSECTION|LR_LOADFROMFILE);
63          if(m_fkBmp == NULL)
64              {
65                  if(MessageBox("缺少图片资源!",
66                      "错误", MB_ICONERROR|MB_OK) == IDOK)
67                  {
68                      //调用退出函数
69                      CDialog::OnCancel();
70                  }
71              }
72              btn->SetBitmap(m_fkBmp);    //按钮图片
73              btn->ShowWindow(SW_SHOW);   //显示图片
74          }
75          else
76          {
77              btn->ShowWindow(SW_HIDE);
78          }
```

```
79          }
80      }
81      //截获用户消息
82      BOOL CLlkDlg::PreTranslateMessage(MSG* pMsg)
83      {
84          //键盘按下消息
85          if(pMsg->message==WM_KEYDOWN)
86          {
87              switch((int)pMsg->wParam)
88              {
89              case VK_F5:                         //快捷键 F5
90                  CallHint();                     //调用查找函数
91                  m_line.m_lineNum = 0;
92                  RefreshMap();                   //更新显示
93                  m_line.Invalidate();
94                  break;
95              case VK_F6:                         //快捷键 F6
96                  CallExchange();                 //调用换盘函数
97                  m_line.m_lineNum = 0;
98                  RefreshMap();                   //更新显示
99                  m_line.Invalidate();
100                 break;
101             default:
102                 break;
103             }
104         }
105         return CDialog::PreTranslateMessage(pMsg);
106     }
```

代码解析：第 8 行调用系统函数设置字体。第 17～25 行调用 TextOut()函数将指定的提示字符串显示到窗口上。第 39 行是调用清空棋盘数组的全部数据。第 40～48 行是根据 map 数组中的数据将整个棋盘中的棋子对象加入棋子数组中。第 49～79 行是遍历整个棋子数组，并根据其行列数和类型加载图片并显示，如果是空的则隐藏。第 82～106 行是根据用户在界面上按下快捷键调用相应的查找或者换盘函数。

（4）游戏中要求实现限制时间和最高记录处理功能，所以必须在主对话框类中再添加定时器处理和最高记录处理功能函数。其代码如代码 13.10 所示。

代码 13.10　定时器及记录处理函数实现

```
01      //定时器
02      void CLlkDlg::OnTimer(UINT nIDEvent)
03      {
04          m_timePoint -= 1;
05          if(m_timePoint <= 0)
06          {
07              KillTimer(1);
08
09              int nRet = MessageBox("限 制 时 间 到!",
10                  "闯 关 失 败", MB_YESNO|MB_ICONINFORMATION);
11
12              if(IDYES == nRet)                   //单击了“确定”按钮
13              {
14                  OnStartGame();                  //重新开始游戏
15              }
16              else
17              {
```

```
18                isHighLevel();                //调用最高分记录函数
19                CDialog::OnCancel();          //退出游戏
20            }
21            return;
22        }
23        InvalidateRgn(&m_MsgRgn);
24
25        ShowMsg(&m_MsgRgn);
26    }
27    //超过游戏最高记录等级
28    void CLlkDlg::isHighLevel()
29    {
30        if(m_nHighLevel<m_nLevel)
31        {
32            CHeroDlg dlg;
33            dlg.SetWriteFlg(TRUE);            //设置写标志
34            dlg.DoModal();                    //弹出英雄榜对话框
35        }
36    }
```

代码解析：第 5 行在定时响应函数中，根据当前时间剩余数判断，如果小于 0 则说明时间已到，提示用户游戏结束，并接收用户是否重新开始的选择，然后根据选择重新开始游戏或者结束游戏。如果是结束游戏，还需要调用超记录信息对话框。

（5）实现主对话框类中棋盘换盘、棋子连接提示及棋盘初始化功能函数。其代码如代码 13.11 所示。

代码 13.11　换盘和连接提示功能函数实现

```
01    //换盘接口成员函数
02    void CLlkDlg::CallExchange()
03    {
04        if(m_exchangeNum)
05        {
06            do
07            Exchange(map);                        //随机更换棋盘数据
08            while(!CChessMan::Hint(map));         //至少有一个可消除
09            ShowMap(map);                         //显示棋盘
10            m_exchangeNum--;                      //换盘记录数减 1
11        }
12    }
13    //提示连接接口成员函数
14    void CLlkDlg::CallHint()
15    {
16        if(m_hintNum)
17        {   //判断两个点是否相连
18            if(CChessMan::Hint(map, &m_hintP1, &m_hintP2))
19            {
20                CChessMan *temp;
21                temp = (CChessMan *)m_cmGroup.GetAt(m_hintP1);
22                temp->SetButtonStyle(BS_DEFPUSHBUTTON);
23                temp = (CChessMan *)m_cmGroup.GetAt(m_hintP2);
24                temp->SetButtonStyle(BS_DEFPUSHBUTTON);
25                m_hintNum--;
26            }
27        }
28    }
29    //换盘函数
```

```
30  void CLlkDlg::Exchange(int map[][MAXY])
31  {
32      int i, j;
33      int k = 0;
34      int temp[(MAXX-2)*(MAXY-2)+1] = {0};     //比格子数多一个，防止数组越界
35      for(i=1; i<MAXX-1; i++)
36      {
37          for(j=1; j<MAXY-1; j++)
38          {
39              if(map[i][j])
40              {
41                  temp[k++] = map[i][j];       //把棋子数据放到临时数组中
42                  map[i][j] = 0;               //清空棋盘数组
43              }
44          }
45      }
46      srand((unsigned int)time(NULL));         //初始化随机数种子发生器
47      k = 0;
48      while(temp[k])
49      {
50          i = rand()%MAXX;                     //随机生成 X 和 Y 坐标
51          j = rand()%MAXY;
52          if(map[i][j])
53              k--;
54          else
55              map[i][j] = temp[k];             //设置当前棋盘数组中的棋子
56          k++;
57      }
58  }
59  //更新棋盘数组
60  void CLlkDlg::RefreshMap()
61  {
62      CRgn resultRgn;                          //创建绘图区域
63      resultRgn.CreateRectRgn(90,30,690,430);
64      CRgn tmpRgn;
65      tmpRgn.CreateRectRgn(0,0,0,0);
66      for(int i=0; i<MAXX; i++)
67      {
68          for(int j=0; j<MAXY; j++)
69          {
70              CRect rect;
71              CRgn srcRgn;
72              if(map[i][j]==-1 || map[i][j]==0)    //当为空时，进行该操作
73              {
74                  rect.top = 30 + i*40;
75                  rect.bottom = 70 + i*40;
76                  rect.left = 90 + j*40;
77                  rect.right = 130 + j*40;
78
79                  srcRgn.CreateRectRgnIndirect(&rect);
80                  //把消去的区域加进来（覆盖原有区域），以便刷新
81                  resultRgn.CombineRgn(&srcRgn,&tmpRgn,RGN_OR);
82                  tmpRgn.CopyRgn(&resultRgn);
83              }
84          }
85      }
86
87      this->InvalidateRgn(&resultRgn);     //只刷新被消去的区域，防止界面闪烁
88  }
```

```
89  //初始化棋盘数组
90  void CLlkDlg::InitMap(int map[][MAXY])
91  {
92      int i, j;
93      int x, y;
94
95      for(i=0; i<MAXX; i++)                   //初始化边界
96      {
97          for(j=0; j<MAXY; j++)
98          {
99              if((i==0) || (i==MAXX-1) || (j==0) || (j==MAXY-1))
100             {
101                 map[i][j] = -1;
102             }
103             else
104             {
105                 map[i][j] = 0;
106             }
107         }
108     }
109     srand((unsigned int)time(NULL));        //随机数种子
110     for(i=0; i<SAME; i++)                   //初始化地图
111     {
112         for(j=1; j<=TYPENUM; j++)
113         {
114             x = rand()%MAXX;
115             y = rand()%MAXY;
116             if(map[x][y])
117             {
118                 j--;
119             }
120             else
121             {
122                 map[x][y] = j;
123             }
124         }
125     }
126 }
```

代码解析：第 6～9 行是调用换盘函数进行棋盘的重新排列，其中使用了循环语句来判断，因为棋子全部是随机排列，有可能还是有无法消除的棋子，所以必须判断有棋子可消除时（调用配对函数 Hint()）才能换盘成功。

代码第 14～28 行是显示连接提示函数的实现，其主要是调用配对函数 Hint()，得到当前棋盘中可以连接的两个棋子，并将其设置为按下状态，这样就实现了提示。第 30～58 行是换盘函数的实现，主要思想是先将整个棋盘数据保存到临时数组中，然后利用随机数重新排列棋盘数组中的数据，实现换盘。

代码第 60～88 行是更新棋盘函数的实现，主要思想是遍历整个棋盘数组，然后将其中为空或者消除后的区域加入到刷新范围内进行刷新，不进行全界面刷新防止闪烁。第 90～126 行是初始化棋盘数组，主要思想是随机将相同类型的 4 个棋子放到棋盘数组不同的位置中。

13.6.3　棋子类的设计

棋子类，包含如下几个功能函数。

1．游戏胜负判断处理

游戏胜负判断处理步骤如下：

（1）等待玩家的鼠标输入信息。

（2）在定时器中，每次扣除游戏的限制时间 1s。

（3）当游戏时间为 0 时，弹出游戏失败结束提示，并根据玩家选择重新开始游戏或者结束游戏。

（4）在玩家消除一对棋子后，就对当前棋盘数组进行遍历。如果是空，说明玩家已经全部消除完毕，然后调用升级处理。如果不是空，则返回步骤（1）。

2．游戏升级处理

游戏升级处理步骤如下：

（1）当游戏中的棋子全部被消除完后，弹出升级提示对话框。

（2）当玩家选择继续时，就调用主对话框类的重新开始游戏接口函数。但要把当前游戏等级增加 1 级。

（3）当玩家选择不继续时，直接调用退出函数。

3．查找处理

查找处理实现步骤如下：

（1）遍历整个棋盘数组。

（2）查找一根直线可以连接的一对棋子，如果没有则转下一步；如果有则转至步骤（6）。

（3）查找有一个拐角，即两根直线可以连接的一对棋子。如果没有则转下一步；如果有则转至步骤（6）。

（4）查找有两个拐角的，即三根直线可以连接的一对棋子。如果没有则转下一步；如果有则转至步骤（6）。

（5）如果返回假，表示查找失败。

（6）如果返回真，表示查找成功，并把棋子的位置在输出参数中返回。

4．提示处理

提示处理是通过调用查找函数，把其中可以连接的一对棋子的位置通过输入参数返回给上层调用者，由调用者设置相关棋子为选中状态。

13.6.4　棋子类的实现

棋子类 CChessMan 的实现，分为如下几个步骤：

（1）声明棋子类 CChessMan，其中包含两个构造函数、查找接口函数、判断棋子为空函数及鼠标左键响应函数。其代码如代码 13.12 所示。

代码 13.12　棋子类的声明

```
01  #if !defined(AFX_CHESSMAN_H__)
```

```
02  #define AFX_CHESSMAN_H__
03
04  ////////////////////////////////////////////
05  // CChessMan 棋子类的声明
06  class CChessMan : public CButton        //棋子类继承于 CButton 按钮类
07  {
08  public:
09      CChessMan();                        //默认构造函数
10      CChessMan(int id, POINT pos);       //带参数的构造函数
11
12  public:
13                                          //查找两个拐角能够连接的棋子
14      static BOOL FindTwoCorner(int map[][MAXY],
15              POINT p1, POINT p2, POINT  *cross1, POINT *cross2);
16                                          //查找一个拐角能够连接的棋子
17      static BOOL FindCorner(int map[][MAXY],
18              POINT p1, POINT p2, POINT *cross1);
19                                          //查找没有拐角能够连接的棋子
20      static BOOL FindLine(int map[][MAXY], POINT p1, POINT p2);
21                                          //提示接口函数
22      static BOOL Hint(int map[][MAXY], int* a1, int *a2);
23      static BOOL Hint(int map[MAXX][MAXY]);
                                            //判断棋盘是否为死盘接口函数
24
25      BOOL IsEmpty(int map[][MAXY]);      //判断棋盘是否为空接口函数
26                                          //查找接口函数 1
27      BOOL Find(int map[MAXX][MAXY], POINT p1, POINT p2);
28                                          //查找接口函数 2
29      BOOL Find(int map[MAXX][MAXY], POINT p1, POINT p2,
30              POINT cross1, POINT cross2);
31                                          //查找两个拐角连接棋子接口函数
32      static BOOL FindTwoCorner(int map[MAXX][MAXY], POINT p1, POINT p2);
33                                          //查找一个拐角连接棋子接口函数
34      static BOOL FindCorner(int map[MAXX][MAXY], POINT p1, POINT p2);
35      POINT m_pos;                        //当前选中位置
36      int m_id;
37      virtual ~CChessMan();               //析构函数
38
39  protected:
40      afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
41      DECLARE_MESSAGE_MAP()
42  };
43  #endif
```

（2）添加棋子类的实现，其中包含基本的构造函数、析构函数、鼠标左键响应函数等。其代码如代码 13.13 所示。

代码 13.13　棋子类的基本函数实现

```
01  // CChessMan.cpp 棋子类实现源文件
02
03  #include "stdafx.h"                     //插入头文件
04  #include "llk.h"
05  #include "ChessMan.h"                   //插入类声明头文件
06  #include "LlkDlg.h"
07
08  ///////////////////////////////////////
```

```
09  // CChessMan 类的实现
10  CChessMan::CChessMan()                          //构造函数
11  {
12  }
13
14  CChessMan::CChessMan(int id, POINT pos)         //带参数的构造函数
15  {
16      m_id = id;                                  //初始化变量
17      m_pos = pos;
18  }
19
20  CChessMan::~CChessMan()                         //析构函数
21  {
22  }
23
24  BEGIN_MESSAGE_MAP(CChessMan, CButton)
25      ON_WM_LBUTTONDOWN()                         //鼠标左键响应函数
26  END_MESSAGE_MAP()
27  //查找两个棋子是否可以用一根直线连接，即没有拐角的
28  //鼠标左键按下响应函数
29  void CChessMan::OnLButtonDown(UINT nFlags, CPoint point)
30  {
31      SetButtonStyle(BS_DEFPUSHBUTTON);           //显示黑边框
32      CLlkDlg *parent = (CLlkDlg *)GetParent();
33      //当前棋子指针为空，即没有任何棋子按下时
34      if((parent->m_p==NULL) || (parent->m_p->m_id!=m_id)
35          || ((parent->m_p->m_pos.x==m_pos.x)
36          &&(parent->m_p->m_pos.y==m_pos.y)))
37      {
38          parent->m_p = this;                     //设置把指针指向当前棋子
39          if(parent->m_line.m_lineNum != 0)      //如果连接线不为 0
40          {
41              parent->m_line.m_lineNum = 0;
42              parent->RefreshMap();
43              parent->m_line.Invalidate();
44          }
45      }
46      else
47      {   //查找两个棋子之间是否有有效连接线
48          if(Find(parent->map, parent->m_p->m_pos,
49              m_pos, parent->m_line.m_crossP1,
50              parent->m_line.m_crossP2))
51          {
52                                                  //消除
53          parent->map[parent->m_p->m_pos.x][parent->m_p->m_pos.y] = 0;
54          parent->map[m_pos.x][m_pos.y] = 0;
55                                      //不能用删除，否则重新开始时无法清除原有的
56          parent->m_p->ShowWindow(SW_HIDE);
57          this->ShowWindow(SW_HIDE);
58          parent->RefreshMap();               //更新数组
59          parent->m_line.Invalidate();
60          parent->m_p = NULL;
61          parent->m_timePoint += 3;           //增加时间点
62                                              //重画信息区域
63          parent->InvalidateRgn(&(parent->m_MsgRgn));
64              if(IsEmpty(parent->map))            //是否消完
65              {
66                  int ret = MessageBox("过关啦!要挑战下一关吗?",
```

```
67                          "恭喜你!", MB_ICONINFORMATION|MB_OKCANCEL);
68                      if(ret == IDOK)
69                      {
70                          parent->Start(++(parent->m_nLevel));
71                      }
72                      else
73                      {
74                          for(int i=0; i<(MAXX-2)*(MAXY-2); i++)
75                          {
76                              delete (CChessMan *)parent->m_cmGroup.GetAt(i);
77                          }
78                                                      //结束游戏时，调用等级判断函数
79                          parent->isHighLevel();
80                                                      //退出程序
81                          parent->DestroyWindow();
82                      }
83                  }
84
85                  if(!Hint(parent->map))          //是否无处可消
86                  {
87                      parent->m_exchangeNum++;
88                      parent->CallExchange();
89                      parent->m_method.SetWindowText("无可用消除!自动切换!");
90                  }
91              }
92              else
93              {
94                  parent->m_p = this;
95              }
96          }
```

代码解析：第 34～36 行条件判断实现的主要思想是判断当前是否按下了棋子，并且是否是同一个棋子，如果是两个不同的棋子，则调用 find()函数查找之间是否可以消除。第 53～63 行实现棋子消除功能，主要是将两个棋子设置为 0 进行隐藏，并更新棋子数组及信息区。

第 64 行是调用 IsEmpty()函数判断棋子是否全部消除完毕，当全部消除完毕时，就给出提示窗口，让用户选择是否继续下一关，用户选择继续则增加等级数，同时调用 start()函数重新开始；用户选择结束的话，则进行最高等级判断及清除窗口工作。

第 85 行是调用 Hint()函数查找棋盘中是否有可以配对的棋子，如果有，则不做任何操作；否则自动调用换盘函数进行换盘，并增加换盘数。

第 94 行是当第 2 个棋子选择后，如果与第 1 个棋子没有有效连线，则将第 2 个棋子设置为当前按下棋子。

（3）实现棋子类的查找接口、棋盘判空功能函数。其代码如代码 13.14 所示。

代码 13.14　查找接口及判空函数实现

```
01  //查找接口函数，输入两个棋子位置
02  BOOL CChessMan::Find(int map[MAXX][MAXY], POINT p1, POINT p2)
03  {
04      CLlkDlg *parent = (CLlkDlg *)GetParent();
05      CString str;
06      if(FindLine(map, p1, p2))
07      {                                       //一根直线可以连接
08          str.Format("直线: (%d,%d)->(%d,%d)\n",
```

```
09                  p1.x, p1.y, p2.x, p2.y);
10              parent->m_method.SetWindowText(str);
11              return TRUE;
12          }
13          if(FindCorner(map, p1, p2))
14          {                                   //一个拐角可以连接
14              str.Format("一拐角: (%d,%d)->(%d,%d)\n",
15                  p1.x, p1.y, p2.x, p2.y);
16              parent->m_method.SetWindowText(str);
17              return TRUE;
18          }
19          if(FindTwoCorner(map, p1, p2))
20          {                                   //两个拐角可以连接
21              str.Format("两拐角: (%d,%d)->(%d,%d)\n",
22                  p1.x, p1.y, p2.x, p2.y);
23              parent->m_method.SetWindowText(str);
24              return TRUE;
25          }
26
27          return FALSE;                       //没有可以连接的棋子
28      }
29      //查找两个棋子间的连接,输出棋子位置
30      BOOL CChessMan::Find(int map[][MAXY],
31                           POINT p1, POINT p2,
32                           POINT cross1, POINT cross2)
33      {
34          CLlkDlg *parent = (CLlkDlg *)GetParent();
35          parent->m_line.m_src1 = p1;
36          parent->m_line.m_src2 = p2;
37
38          CString str;
39          if(FindLine(map, p1, p2))
40          {                                   //一根直线可以连接
41              str.Format("直线: (%d,%d)->(%d,%d)\n",
42                  p1.x, p1.y, p2.x, p2.y);
43              parent->m_method.SetWindowText(str);
44              parent->m_line.m_lineNum = 1;
45              return TRUE;
46          }
47          if(FindCorner(map, p1, p2, &(parent->m_line.m_crossP1)))
48          {                                   //两根直线可以连接
49              str.Format("一拐角: (%d,%d)->(%d,%d)\n",
50                  p1.x, p1.y, p2.x, p2.y);
51              parent->m_method.SetWindowText(str);
52              parent->m_line.m_lineNum = 2;
53              return TRUE;
54          }
55          if(FindTwoCorner(map, p1, p2, &(parent->m_line.m_crossP1),
56              &(parent->m_line.m_crossP2)))
57          {                                   //三根直线可以连接
58              str.Format("两拐角: (%d,%d)->(%d,%d)\n",
59                  p1.x, p1.y, p2.x, p2.y);
60              parent->m_method.SetWindowText(str);
61              parent->m_line.m_lineNum = 3;
62              return TRUE;
63          }
64
65          return FALSE;                       //没有棋子可以连接
66      }
```

```
67   //判断棋盘上的棋子是否被消除完毕
68   BOOL CChessMan::IsEmpty(int map[][MAXY])
69   {
70       int i, j;
71       int sum = 0;
72   
73       for(i=1; i<MAXX-1; i++)
74           for(j=1; j<MAXY-1; j++)
75               sum += map[i][j];
76       if(sum)
77           return FALSE;                      //没有消除完
78       else
79           return TRUE;                       //消除完毕
80   }
```

代码解析：第 2 行是查找函数的实现，主要是利用输入两个棋子坐标来查找有无连线。第 30 行是另一种查找函数，是在前一种基础上增加了棋子间有连接就要把棋子的拐角位置输出。

（4）实现棋子类的配对接口函数分为两种：一种是判断当前棋盘上有无配对棋子；另一种是在前一种的基础上添加输出配对棋子坐标的功能，其代码如代码 13.15 所示。

代码 13.15　配对接口函数实现

```
01   //判断当前棋盘上有没有可以配对的棋子
02   BOOL CChessMan::Hint(int map[][MAXY])
03   {
04       int x1, y1, x2, y2;                //初始化临时变量
05       POINT p1 = {0};                    //初始化点信息
06       POINT p2 = {0};                    //初始化点信息
07       for(x1=1; x1<MAXX-1; x1++)
08       {
09           for(y1=1; y1<MAXY-1; y1++)
10           {
11               for(x2=1; x2<MAXX-1; x2++)
12               {
13                   for(y2=1; y2<MAXY-1; y2++)
14                   {
15                       p1.x = x1;
16                       p1.y = y1;
17                       p2.x = x2;
18                       p2.y = y2;
19                       if((map[x1][y1]==0) || (map[x2][y2]==0))
20                       {                  //空白格子
21                           continue;
22                       }
23                       if(map[x1][y1] != map[x2][y2])
24                       {                  //不相等
25                           continue;
26                       }
27                       if((x1==x2) && (y1==y2))
28                       {                  //同一个点
29                           continue;
30                       }
31                       if(FindLine(map, p1, p2))
32                       {                  //一根直线可以连接
33                           return TRUE;
34                       }
```

```
35                      if(FindCorner(map, p1, p2))
36                      {                   //一个拐角可以连接
37                          return TRUE;
38                      }
39                      if(FindTwoCorner(map, p1, p2))
40                      {                   //两个拐角可以连接
41                          return TRUE;
42                      }
43                  }
44              }
45          }
46      }
47      return FALSE;                       //没有可以配对的棋子
48  }
49  //查找棋盘上可以配对的棋子，成功时，输出棋子位置
50  BOOL CChessMan::Hint(int map[][MAXY], int *a1, int *a2)
51  {
52      int x1, y1, x2, y2;                 //初始化临时变量
53      POINT p1 = {0};                     //初始化点信息
54      POINT p2 = {0};                     //初始化点信息
55
56      for(x1=1; x1<MAXX-1; x1++)          //循环遍历数组
57      {
58          for(y1=1; y1<MAXY-1; y1++)
59          {
60              for(x2=1; x2<MAXX-1; x2++)
61              {
62                  for(y2=1; y2<MAXY-1; y2++)
63                  {
64                      p1.x = x1;
65                      p1.y = y1;
66                      p2.x = x2;
67                      p2.y = y2;
68                      if((map[x1][y1]==0) || (map[x2][y2]==0))
69                      {                   //空白格子
70                          continue;
71                      }
72                      if(map[x1][y1] != map[x2][y2])
73                      {                   //不相等
74                          continue;
75                      }
76                      if((x1==x2) && (y1==y2))
77                      {                   //同一个点
78                          continue;
79                      }
80                      if(FindLine(map, p1, p2))
81                      {                   //一根直线可连接
82                          *a1 = (x1-1) * (MAXY-2) + y1 - 1;
83                          *a2 = (x2-1) * (MAXY-2) + y2 - 1;
84                          return TRUE;
85                      }
86                      if(FindCorner(map, p1, p2))
87                      {                   //两根直线可连接
88                          *a1 = (x1-1) * (MAXY-2) + y1 - 1;
89                          *a2 = (x2-1) * (MAXY-2) + y2 - 1;
90                          return TRUE;
91                      }
92                      if(FindTwoCorner(map, p1, p2))
93                      {                   //三根直线可连接
```

```
94                          *a1 = (x1-1) * (MAXY-2) + y1 - 1;
95                          *a2 = (x2-1) * (MAXY-2) + y2 - 1;
96                          return TRUE;
97                      }
98                  }
99              }
100         }
101     }
102     return FALSE;                       //查不到可以连接的棋子
103 }
```

代码解析：第 2 行是判断当前棋盘上有没有可以配对的棋子的函数，实现的主要思想是遍历全部棋盘数组中的元素，但要除去其中棋子类型为消除了的（0）或者类型不相同及棋子位置相同的。然后查找棋盘数组中的元素。查找两个棋子之间有无可以有效连接的线，如果有，则说明还有可配对的棋子。

第 50 行同样是判断当前棋盘上有没有可以配对的棋子的函数，其不同是需要输出两个棋子的位置，这样就可以将两个棋子显示出来。

（5）实现各种不同的查找函数。包含查找一根直线连接函数、查找一个拐角连接函数和查找两个拐角连接函数。其代码如代码 13.16 所示。注意，这里只给出了其中一种没有输出拐角坐标的函数实现，另一种输出拐角坐标的函数实现请读者查阅本书配套光盘上的源文件。

代码 13.16　各种查找函数实现

```
01 //查找一根直线
02 BOOL CChessMan::FindLine(int map[][MAXY], POINT p1, POINT p2)
03 {
04     int max, min;                           //定义最大最小变量
05     int i;                                  //定义计数变量
06 
07     if(p1.x == p2.x)                        //判断两个棋子是同一行的
08     {
09         max = (p1.y>p2.y)?p1.y:p2.y;
10         min = (p1.y<p2.y)?p1.y:p2.y;
11         if(max == min+1)
12         {
13             return TRUE;                    //相邻的两个格子
14         }
15         for(i=min+1; i<max; i++)
16         {
17             if((map[p1.x][i]!=0) && (map[p1.x][i]!=-1))
18             {
19                 return FALSE;               //两个棋子不可连
20             }
21         }
22         return TRUE;                        //两个棋子可连
23     }
24 
25     if(p1.y == p2.y)                        //判断两个棋子是同一列的
26     {
27         max = (p1.x>p2.x)?p1.x:p2.x;
28         min = (p1.x<p2.x)?p1.x:p2.x;
29         if(max == min+1)
30             return TRUE;                    //相邻的两个格子
31         for(i=min+1; i<max; i++)
```

```
32              if((map[i][p1.y]!=0) && (map[i][p1.y]!=-1))
33              {
34                  return FALSE;               //两个棋子不可连
35              }
36          return TRUE;                        //两个棋子可连
37      }
38      return FALSE;                           //两个棋子不可连
39  }
40  //查找两个棋子是否用两根直线可以连接，即一个拐角的
41  BOOL CChessMan::FindCorner(int map[][MAXY], POINT p1, POINT p2)
42  {
43      int maxx, maxy, minx, miny;
44      POINT tempPoint = {0};
45
46      maxx = (p1.x>p2.x)?p1.x:p2.x;           //测试矩形格子
47      maxy = (p1.y>p2.y)?p1.y:p2.y;
48      minx = (p1.x<p2.x)?p1.x:p2.x;
49      miny = (p1.y<p2.y)?p1.y:p2.y;
50      if(map[minx][maxy] == 0)                //判断矩形格子的各个角是否有棋子
51      {
52          tempPoint.x = minx;
53          tempPoint.y = maxy;
54          if((FindLine(map, p1, tempPoint))
55              &&(FindLine(map, tempPoint, p2)))
56              return TRUE;                    //可以连接
57      }
58
59      if(map[maxx][miny] == 0)                //判断矩形格子的各个角是否有棋子
60      {
61          tempPoint.x = maxx;
62          tempPoint.y = miny;
63          if((FindLine(map, p1, tempPoint))
64              && (FindLine(map, tempPoint, p2)))
65              return TRUE;                    //可以连接
66      }
67
68      if(map[minx][miny] == 0)                //判断矩形格子的各个角是否有棋子
69      {
70          tempPoint.x = minx;
71          tempPoint.y = miny;
72          if((FindLine(map, p1, tempPoint))
73              && (FindLine(map, tempPoint, p2)))
74              return TRUE;                    //可以连接
75      }
76
77      if(map[maxx][maxy] == 0)                //判断矩形格子的各个角是否有棋子
78      {
79          tempPoint.x = maxx;
80          tempPoint.y = maxy;
81          if((FindLine(map, p1, tempPoint))
82              && (FindLine(map, tempPoint, p2)))
83              return TRUE;                    //可以连接
84      }
85      return FALSE;                           //不可以连接
86  }
87  //查找三根直线可以连接的棋子，即两个拐角的
88  BOOL CChessMan::FindTwoCorner(int map[][MAXY], POINT p1, POINT p2)
89  {
90      int i;
```

```
91      POINT tempPoint = {0};
92
93      for(i=0; i<MAXY; i++)                    //纵向遍历
94      {
95          if(i == p1.y)
96          {
97              continue;
98          }
99          tempPoint.x = p1.x;
100         tempPoint.y = i;
            //必须是消除掉的点或者边界
101         if((map[tempPoint.x][tempPoint.y]==0)
102             || (map[tempPoint.x][tempPoint.y]==-1))
103             if(FindLine(map, tempPoint, p1))
104             {
105                 tempPoint.x = p2.x;
106                 if((map[tempPoint.x][tempPoint.y]==0)
107                     || (map[tempPoint.x][tempPoint.y]==-1))
108                     if(FindLine(map, tempPoint, p2))
109                         return TRUE;     //查到可以连接
110             }
111     }
112
113     for(i=0; i<MAXX; i++)                    //纵向遍历
114     {
115         if(i == p1.x)
116             continue;
117         tempPoint.x = i;
118         tempPoint.y = p1.y;
            //必须是消除掉的点或者边界
119         if((map[tempPoint.x][tempPoint.y]==0)
120             || (map[tempPoint.x][tempPoint.y]==-1))
121             if(FindLine(map, tempPoint, p1))
122             {
123                 tempPoint.y = p2.y;
124                 if((map[tempPoint.x][tempPoint.y]==0)
125                     || (map[tempPoint.x][tempPoint.y]==-1))
126                     if(FindLine(map, tempPoint, p2))
127                         return TRUE;     //查到可以连接的
128             }
129     }
130     return FALSE;                            //没有可以连接的
131 }
...                                          //省略部分代码，请查阅光盘源文件
```

代码解析：第 2 行是查找两个棋子之间是否有直接连接的实现，实现思想是先判断是否是同一行或者同一列，如果是同一行（即代码第 7 行的判断），则比较两个棋子同一行间是否有棋子存在（0 和－1 不算，0 是已消除，－1 为边界），如果有，则说明不可连。否则说明可连。如果是同一列（即代码第 25 行的判断），也是采用同样的判断。

第 41 行是查找两个棋子之间是否有 2 条直线可以连接的实现，实现思想是查找由两个棋子所在点构成的矩形的各个角是否有棋子存在，如果有，就直接跳过；没有才判断该点与两个棋子分别是不是可以用直接相连，如果可相连，则说明两个棋子用 2 条直线可以连接，反之不可连。最后如果各个角都不为空，则说明两个棋子不可连。

第 88 行是查找三根直线可以连接的棋子的实现，实现思想是在整个棋盘数组中查找 2（a、b）个点（这两个点必须是消除过的棋子或者是边界位置）。其中 a 点能够与棋子 1 直

线相连，b 点能够与棋子 2 直线相连，那么两个棋子是配对的，它们就可以被消除掉。

13.7　连连看游戏的整合测试

在本节中，笔者将根据前面的测试用例文档进行连连看游戏的整合测试。在这里，笔者只对其中几个主要的测试用例进行演示。

13.7.1　主菜单和界面显示功能测试的演示

这个测试用例主要是测试游戏的菜单和界面显示是否成功，根据测试用例文档，其测试步骤如下所述。

（1）运行连连看程序，选中其中的 llk.exe 图标，如图 13.9 所示。



图 13.9　运行连连看程序

（2）程序启动后，其菜单及主界面如图 13.10 所示。



图 13.10　连连看游戏主界面及菜单

判断结果：游戏的菜单和界面显示成功。填写测试用例编号 1.7.1 结果为“通过”。

13.7.2　消除相同棋子功能测试的演示

消除相同棋子功能测试，根据测试用例文档，其测试步骤如下所示。

（1）游戏开始后，查看其中相同的棋子，选中其中一个，如图 13.11 所示。

（2）选中两个相同的棋子，棋子被消除，如图 13.12 所示。

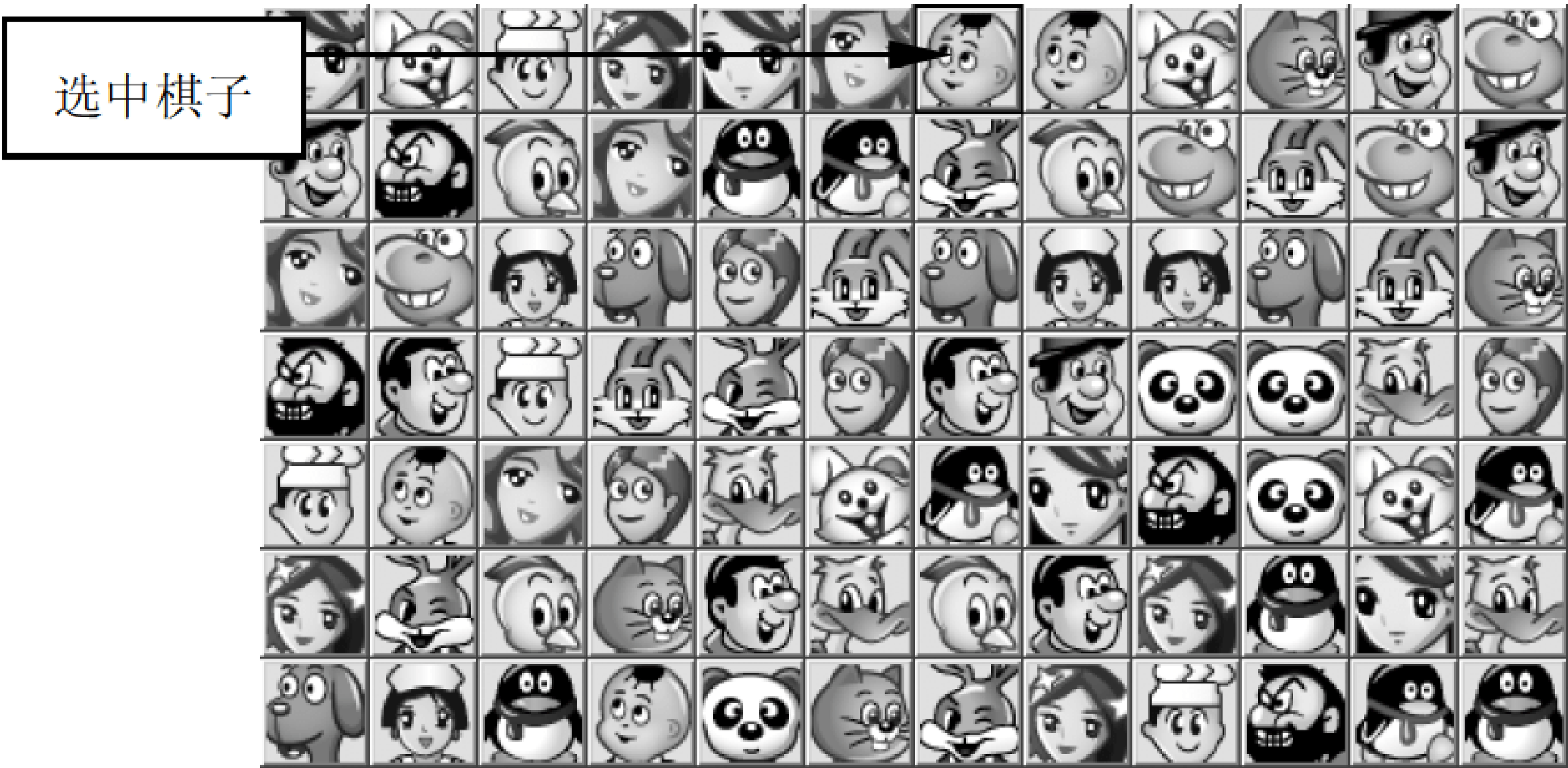


图 13.11　选中相同的棋子

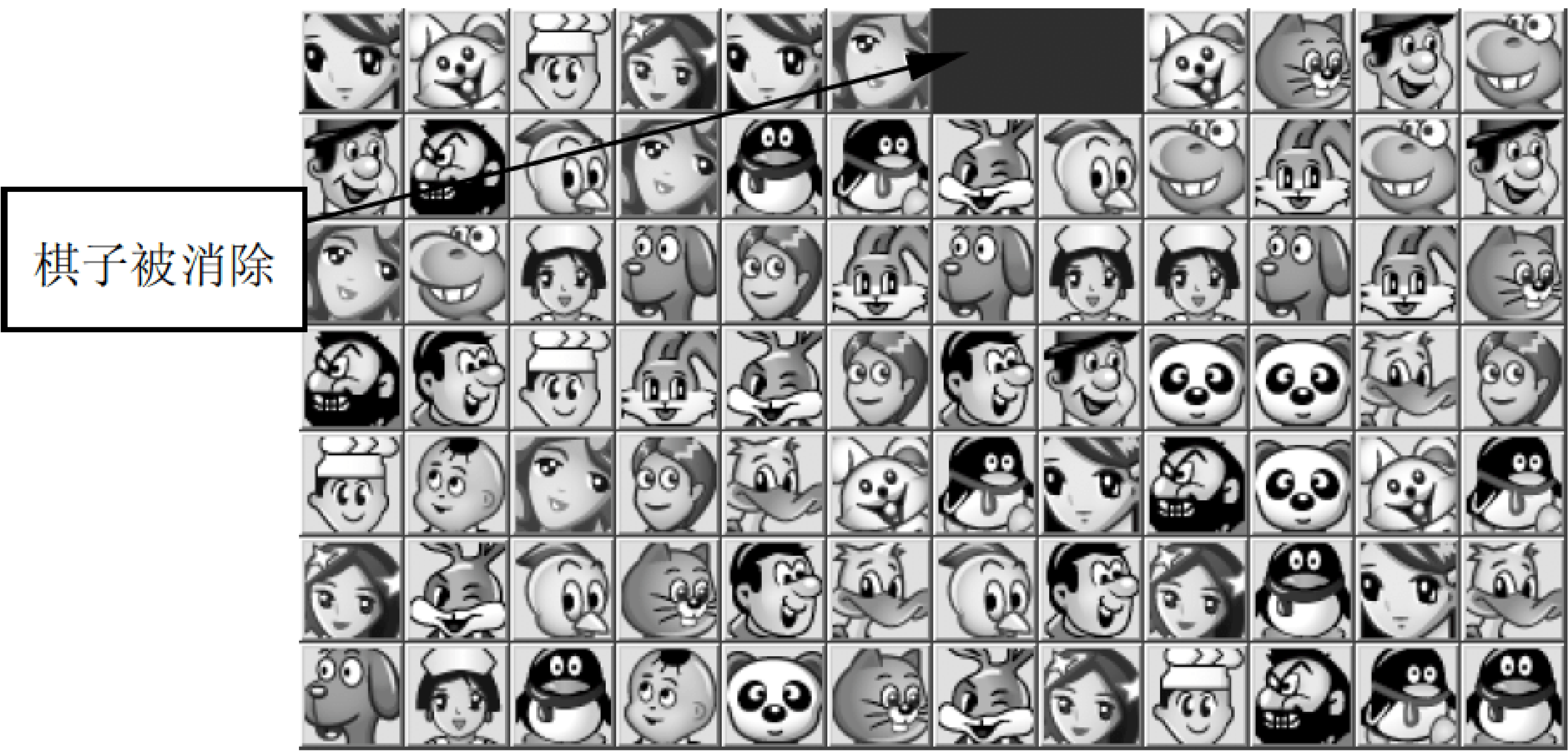


图 13.12　选中相同的棋子，棋子被消除

判断结果：消除相同棋子功能测试成功。填写测试用例编号 1.7.2 结果为“通过”。

13.7.3　游戏升级功能测试的演示

测试游戏中的升级功能。根据测试用例文档，其测试步骤如下所述。

（1）查看当前游戏等级，如图 13.13 所示。

（2）消除完全部棋子时，查看有无提示。单击“确定”按钮，如图 13.14 所示。

（3）升级后，查看当前游戏等级，如图 13.15 所示。



图 13.13　当前游戏等级



图 13.14　消除完全部棋子时



图 13.15　升级后游戏等级

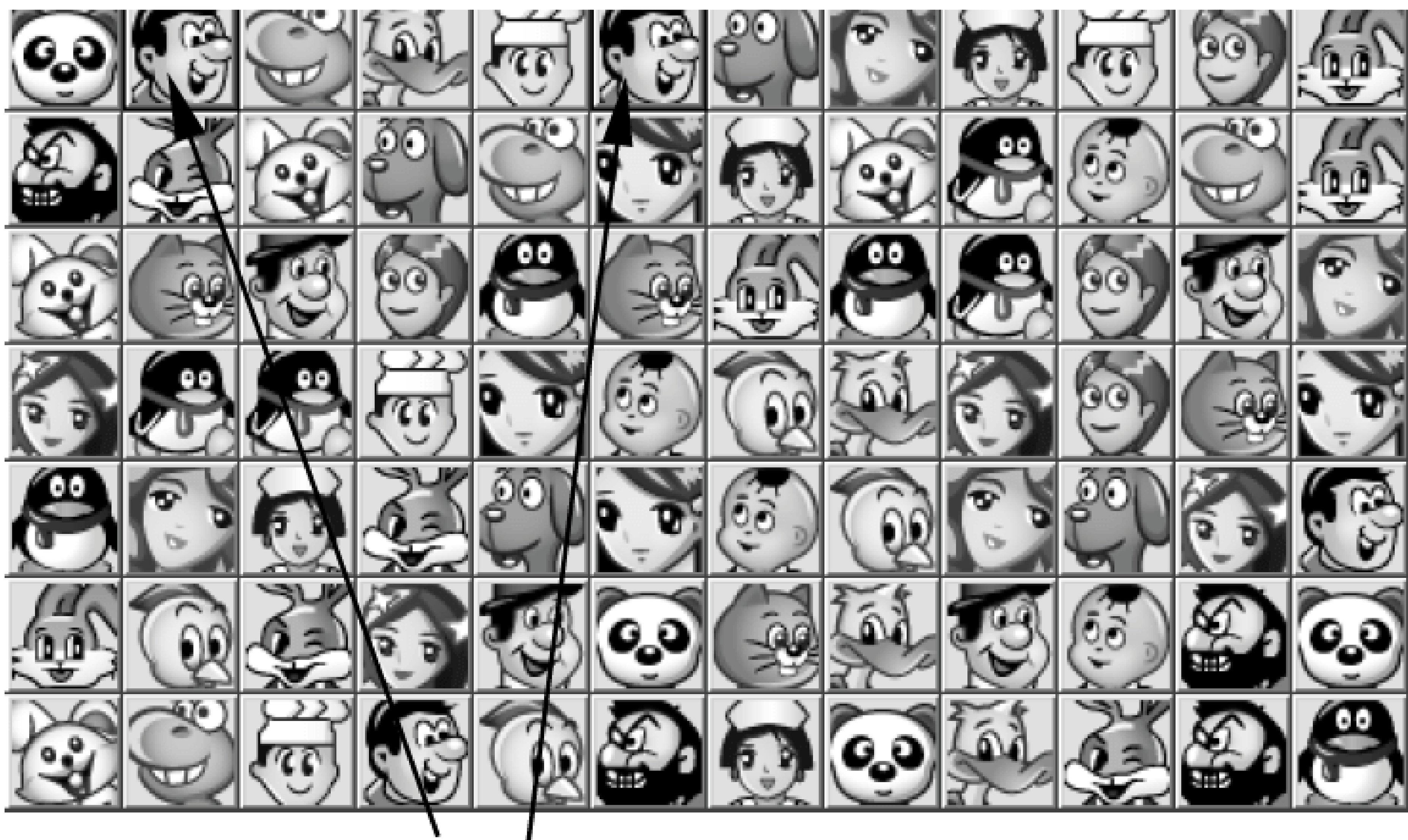
判断结果：通过比较，游戏中的升级功能正确。填写测试用例编号 1.7.4 结果为“通过”。

13.7.4　消除提示功能测试的演示

测试连连看游戏中使用快捷键 F5，能否进行消除提示。根据测试用例文档，其测试步骤如下所述。

（1）按下键盘上的快捷键 F5，调用消除提示功能。

（2）游戏界面上有配对棋子提示，如图 13.16 所示。



同时出现选中状态，作为提示

图 13.16　消除配对提示

判断结果：连连看游戏支持消除配对提示。填写测试用例编号 1.7.8 结果为“通过”。

13.7.5　游戏帮助功能测试的演示

测试连连看游戏是否有帮助提示功能。根据测试用例文档，其测试步骤如下所述。

（1）选择“帮助”|“帮助”命令，如图 13.17 所示。

（2）游戏中弹出帮助对话框，如图 13.18 所示。

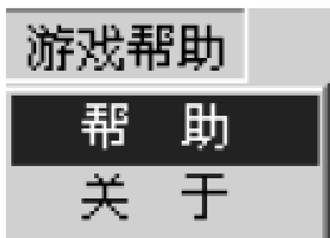


图 13.17　选择“游戏帮助”|“帮助”命令

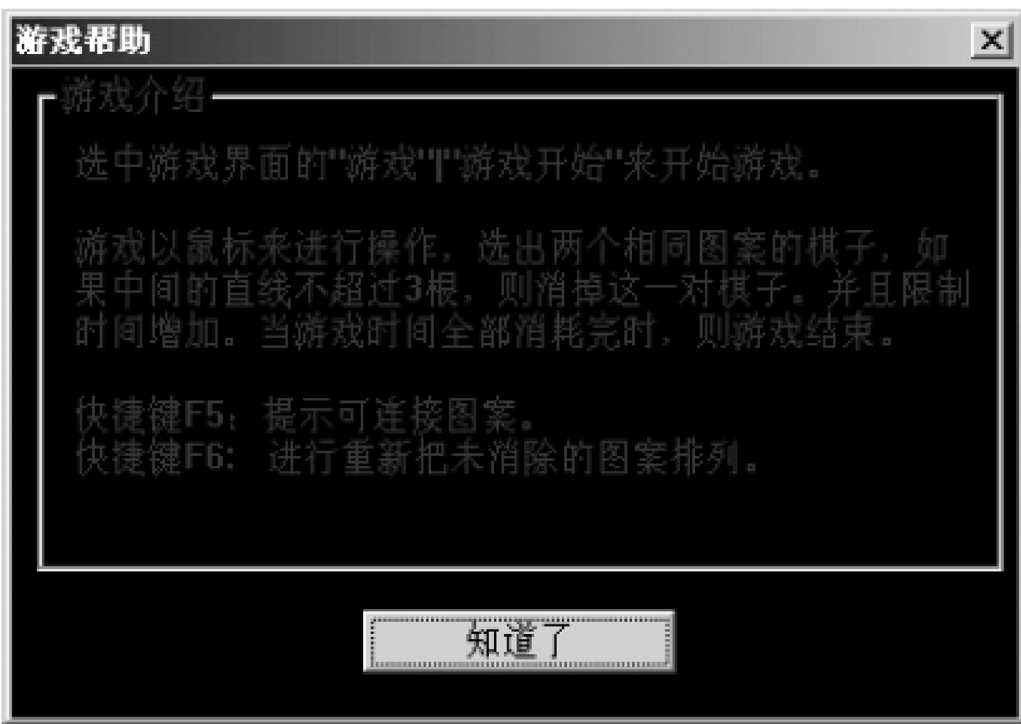


图 13.18　弹出“游戏帮助”对话框

判断结果：连连看游戏帮助提示是正确的。填写测试用例编号 1.7.6 结果为“通过”。

13.7.6　棋子换盘功能测试的演示

测试游戏中的棋子换盘功能是否正确，根据测试用例文档，其测试步骤如下所述。

（1）运行游戏后，消除若干配对棋子，如图 13.19 所示。

（2）按下快捷键 F6，调用换盘功能后，棋盘如图 13.20 所示。



图 13.19　消除若干棋子的棋盘



图 13.20　换盘后的棋盘

判断结果：通过比较，认为连连看的棋子换盘功能是正确的。填写测试用例编号 1.7.9 结果为“通过”。

13.8　总　　结

通过本章连连看游戏实例项目开发的学习，希望读者能够更加深入地掌握好游戏项目开发中各种文档的格式及编写方法。同时，知道自己如何开发一款连连看游戏。本游戏最核心的算法包括：棋盘的初始化、消除提示、换盘功能、直线连接查找、一拐角连接查找、两拐角连接查找等。这些都是本游戏最难理解的部分，请读者朋友采用一边阅读一边实践的方式来学习，这样才能深刻理解整个算法的核心思想。

第 14 章　黑白棋游戏项目开发

学习完第 13 章连连看游戏的开发后，本章将继续介绍黑白棋游戏实例项目的开发。本章的内容主要是为了帮助读者继续增加项目实际开发的经验，提高对各种文档的编写能力。

本章主要涉及的内容如下：

- 黑白棋项目的需求分析。
- 黑白棋游戏的概要设计。
- 黑白棋游戏操作界面设计。
- 黑白棋游戏的详细设计及代码。
- 黑白棋游戏的测试用例文档的编写及测试演示。

14.1　黑白棋游戏项目的需求分析

获得用户需求并对其进行详细分析，是项目开始的基础。只有获得明确的需求，并做出好的需求分析文档得到客户的认可，才能保证项目的成功。

技巧：从几个典型用户处获得的需求，比从大量用户处获得的需求效果更好。

14.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到黑白棋游戏开发的资料如下：

1．黑白棋游戏概述

黑白棋，又叫“Othello 棋”或者“翻转棋”，是 19 世纪末英国人发明的，在西方和日本很流行。直到 20 世纪 70 年代一个日本人将其发展，也就是现在大家玩的黑白棋。游戏通过相互翻转对方的棋子，最后以棋盘上谁的棋子多来判断胜负。其游戏规则比较简单，因此很容易学会，但是变化却又非常复杂。有一种说法是：只需要几分钟学会，但却需要一生的时间去精通。

2．黑白棋的操作方法

玩家把属于自己颜色的棋子放在棋盘的空格上，而当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子，则被夹在中间的全部翻转会成为自己颜色的棋子。

例如，如果玩家 A 执黑棋，并且看到在一排白棋的某一端是一颗黑棋，那么当玩家 A

将一颗黑棋放在这一排的另一端时，所有的白棋都将翻转并变为黑棋。所有的水平、垂直或对角直线方向均有效。

3. 黑白棋游戏的基本规则

黑白棋的棋盘是一个有 8×8 方格的棋盘。下棋时将棋下在空格中间，而不是像围棋一样下在交叉点上。开始时在棋盘正中有两白两黑四个棋子交叉放置，而黑棋总是先下子。走棋的唯一规则是只能走包围并翻转对手的棋子。每一回合都必须至少翻转一颗对手的棋子。如果玩家在棋盘上没有地方可以下子，则该玩家对手可以连下。双方都没有棋子可以下时棋局结束，以棋子数目来计算胜负，棋子多的一方获胜。

在棋盘还没有下满时，如果一方的棋子已经被对方吃光，则棋局也结束。将对手棋子吃光的一方获胜。

4. 能够实现人机对战模式

即其中一方为电脑，另一方为玩家，实现人和电脑的较量，具有一定的人工智能性。有一方胜利时进行统计并给出提示，并且能够设置电脑的游戏等级。

5. 界面的美化

在本游戏中，要求对棋子的翻转变化进行动画的演示。

6. 支持悔棋功能

即在玩家落子错误的情况下，可以进行悔棋操作。

7. 有背景音乐支持

在游戏中，能够通过选择播放背景音乐。

8. 游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

14.1.2　对语言描述进行需求分析

根据《黑白棋用户需求描述文档》中的内容，在需要对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

技巧：对语言描述的分析要功能越简单越好。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类黑白棋游戏。特制定本说明书来用于描述某公司黑白棋项目开发的功能性需求。

1.1　编写目的

使用技术性语言对某公司的黑白棋游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含某公司黑白棋游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司黑白棋游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

《黑白棋用户需求描述文档》。

5．游戏具有的功能

5.1　能够显示主菜单和界面

游戏需要提供主菜单让玩家进行游戏设置，同时能够显示当前黑白棋子数量等相关信息到界面上。

5.2　能够接收鼠标输入功能

能够接收玩家的鼠标输入功能，把棋子放入棋盘上指定的位置。

5.3　能够根据规则翻转相应的棋子

无论是电脑或者是玩家在棋盘中落下了棋子后，能够根据游戏规则，把横向、纵向及对角直线上的棋子全部翻转过来，变成最后落下棋子的颜色，并对棋子对比数量进行增减。

例如，如果玩家 A 执黑棋，并且看到在一排白棋的某一端是一颗黑棋，那么当玩家 A 将一颗黑棋放在这一排的另一端时，所有的白棋都将翻转并变为黑棋。所有的水平、垂直或对角直线方向均有效。

5.4　游戏胜负判断功能

双方都没有棋子可以下时棋局结束，以棋子数目来计算胜负，棋子多的一方获胜。在棋盘还没有下满时，如果一方的棋子已经被对方吃光，则棋局也结束。将对手棋子吃光的一方获胜。

5.5　能够实现人机对战模式

支持人工智能，根据游戏等级的不同，进行不同的算法深度搜索，找出最合理的位置进行落子操作。

5.6　人工智能的等级设置

能够指定当前电脑人工智能的等级。根据不同的等级，其算法搜索深度不同。

5.7　游戏悔棋功能

游戏中支持玩家进行悔棋操作。

5.8　游戏支持背景音乐功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.9　游戏提供帮助说明

在游戏菜单中，提供一个使用说明项。以方便对本游戏不了解的玩家对游戏进行操作和使用。

14.2　黑白棋游戏概要设计

笔者编写的黑白棋游戏的概要设计文档内容如下所述。

1．引言

1.1　编写目的

为了让各个开发人员明白黑白棋游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．术语

3．参考文献

《黑白棋游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高鼠标及键盘操作的易用性。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。
- Microsoft® Visual C++ 6.0。

4.3　需求概述

参见《黑白棋游戏需求分析说明书》。

4.4　条件与限制

无。

5．总体设计

5.1　黑白棋游戏的功能架构（如图 14.1 所示）

5.2　各功能处理流程

内容参见《黑白棋游戏各功能详细设计文档》。

6．接口设计

内容参见《黑白棋游戏操作界面设计文档》。

7．类结构设计

游戏由 5 个类和 1 功能模块组成，如图 14.2 所示。

- 主界面对话框类：主要负责主界面及菜单、棋盘、棋子的显示及棋盘窗口类对象的创建和调用等处理。
- 棋盘窗口类：主要负责接收玩家鼠标输入的棋子位置及棋子翻转动画等处理。

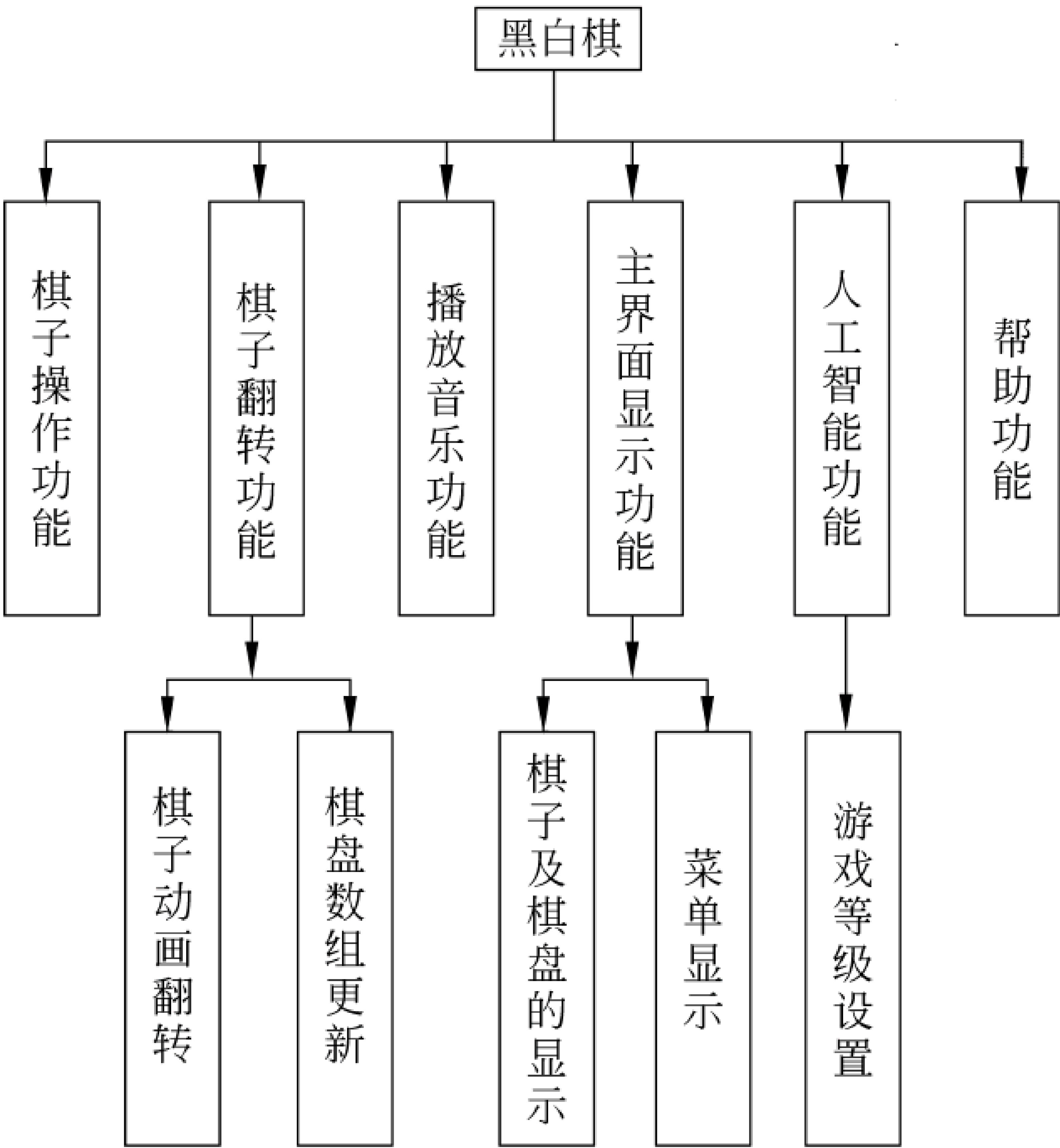


图 14.1　黑白棋功能架构

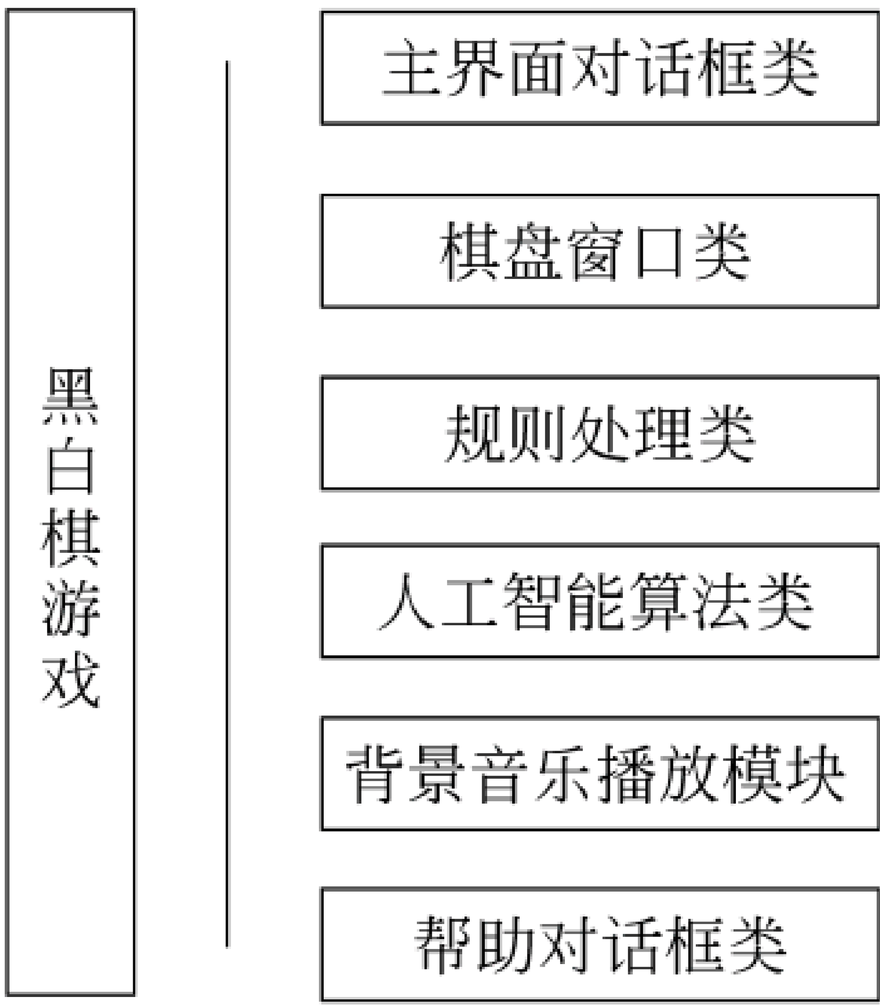


图 14.2　游戏主要类结构

- 规则处理类：主要负责棋子数据的统计、落子位置有效及胜负判断等处理。
- 人工智能算法类：主要负责电脑的人工智能算法处理。
- 背景音乐播放模块：主要负责游戏中背景音乐的播放。
- 帮助对话框类：主要负责帮助提示的显示及其他辅助信息。

8．出错处理设计

8.1　出错输出信息

当游戏中出现错误，采用弹出对话框的方式来提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误，采用中止当前游戏并重新开始游戏的方法来处理游戏中的错误。

9．维护设计

由于整个黑白棋游戏项目在开发完成后，基本不会有太多的变动，所以维护的主要任务是只要把用户使用中出现的错误解决。

14.3　黑白棋游戏操作界面及测试用例设计

本章将继续介绍《游戏操作界面设计文档》和《游戏测试用例文档》的编写。

14.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的黑白棋游戏的操作界面设计文档内容如下所述。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确黑白棋游戏的操作界面是如何设计的，特制定本文档，用于描述本公司黑白棋游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及其客户。

2．文档范围

包含本公司黑白棋游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《黑白棋游戏需求分析说明书》。

《黑白棋游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面的设计

黑白棋的游戏主界面设计，如图 14.3 所示。

5.2　游戏菜单结构的设计

黑白棋的游戏菜单设计如图 14.4 所示。

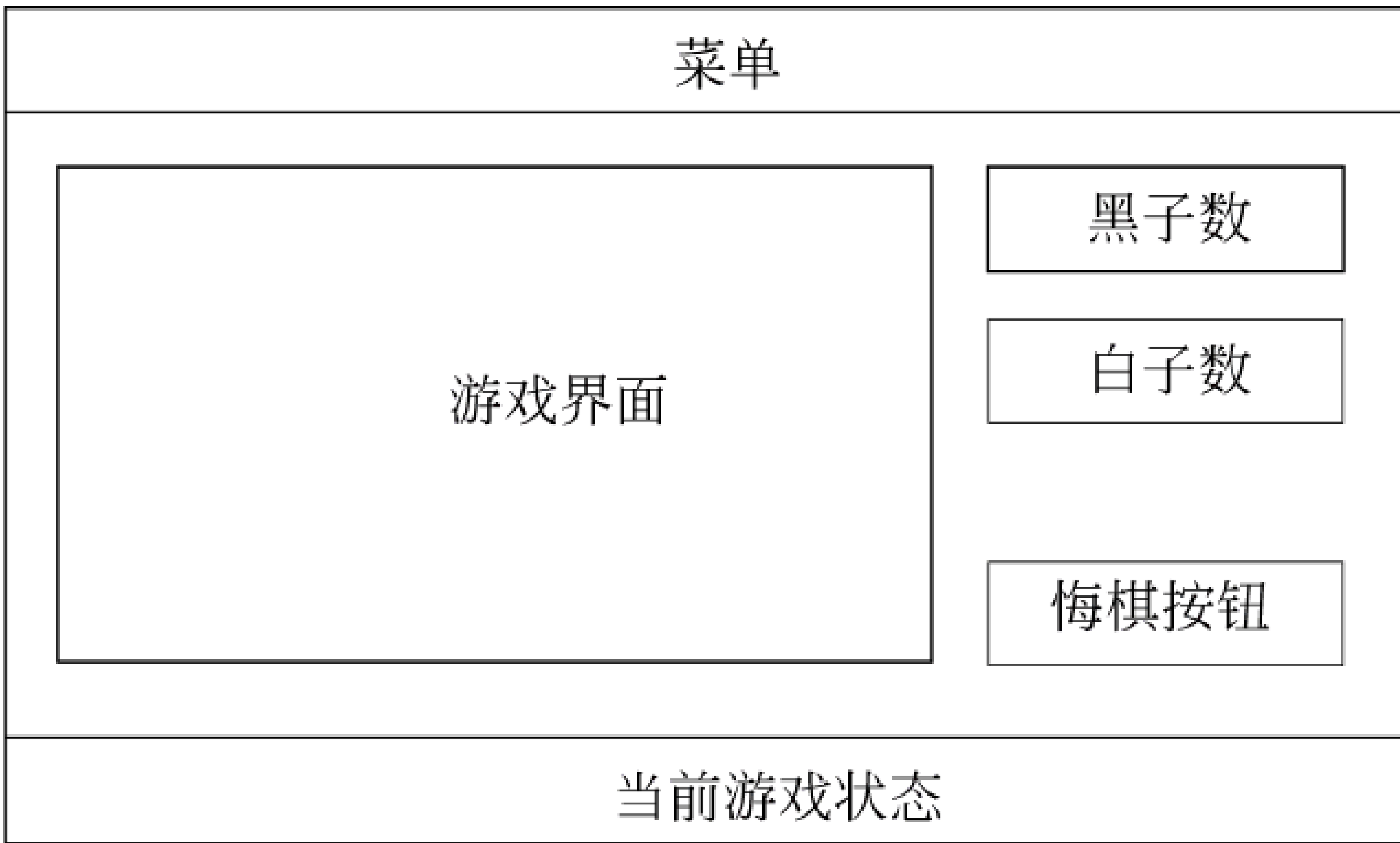


图 14.3　设计的游戏主界面

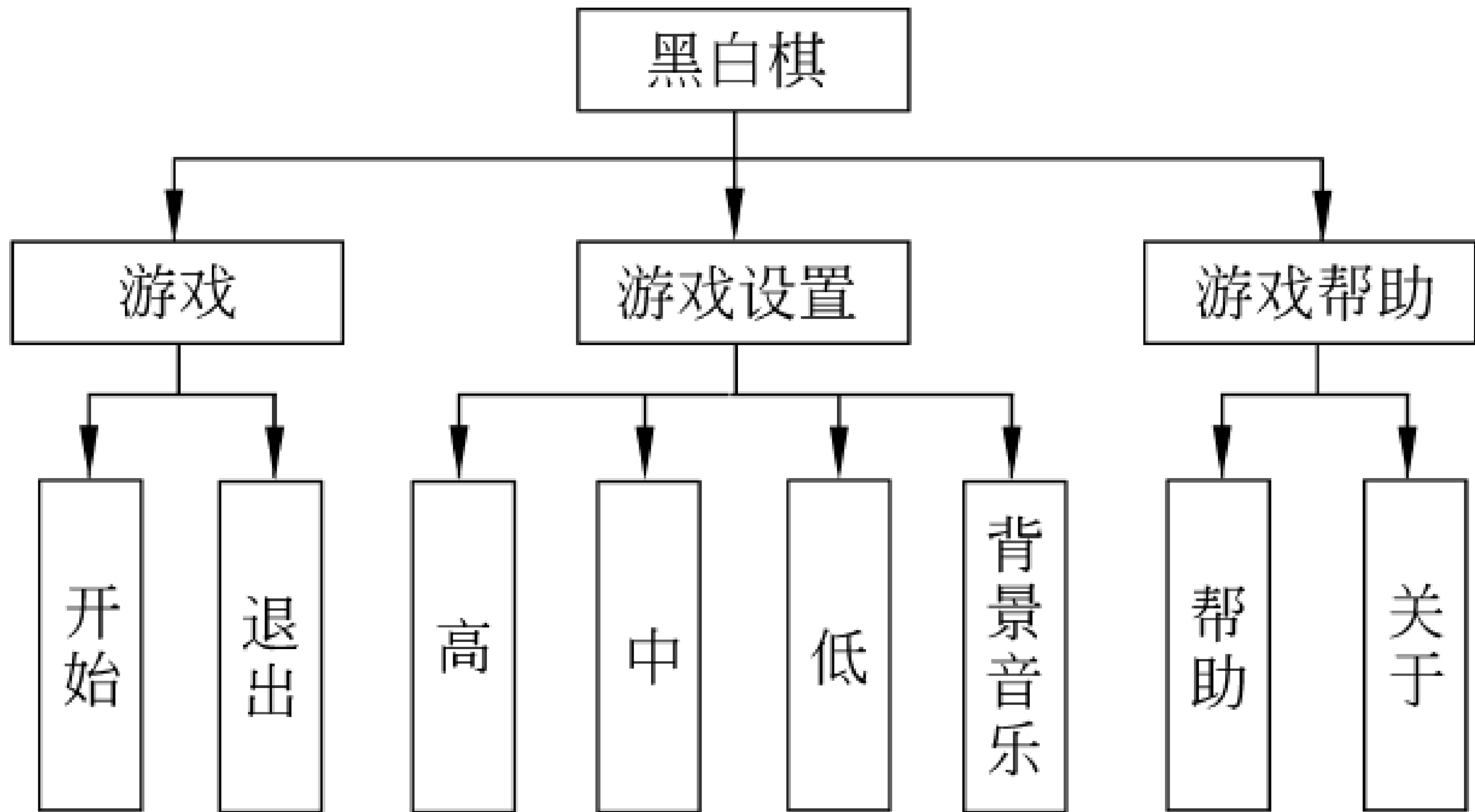


图 14.4　设计的游戏菜单结构

14.3.2　测试用例文档

测试用例文档主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的黑白棋游戏的测试用例文档内容如下所述。

1．引言

本文档主要用于某公司在开展黑白棋游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了黑白棋游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过，方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的黑白棋游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行黑白棋游戏项目的测试和验收。

3．使用对象

本测试用例文档使用对象主要是与某公司黑白棋游戏开发相关的需求分析、测试和维护等部门（单位）的人员。

4．参考文献

《黑白棋游戏的需求分析说明书》；

《黑白棋游戏的概要设计文档》；

《黑白棋游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试项目

测试项目主要针对黑白棋中的各种功能进行整合性测试，共包含如下几个项目。

（1）主菜单和界面显示功能的测试，主要内容如表 14.1 所示。

表 14.1　主菜单和界面显示功能的测试

测试编号：1.7.1	类别：A
项　　目：黑白棋测试	
分 项 目：主菜单和界面显示功能的测试	
测试目的：测试黑白棋游戏中的菜单和界面是否正确显示	
测试配置：	
预置条件： 黑白棋游戏源程序已经编译完成，并可以运行； 键盘和鼠标已准备好	
测试步骤： 运行黑白棋程序，查看菜单和界面	
预期结果： 游戏主界面及菜单与操作设计文档中的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏主界面和菜单是否正确显示（是/否）	
测试结果： 通过/不通过	

（2）悔棋功能的测试，主要内容如表 14.2 所示。

表 14.2　悔棋功能的测试

测试编号：1.7.2	类别：A
项　　目：黑白棋测试	
分 项 目：悔棋功能的测试	
测试目的：测试游戏是否支持悔棋功能	
测试配置：	
预置条件： 黑白棋游戏已经开始	
测试步骤： 与电脑进行多步落子操作； 单击“悔棋”按钮； 查看棋盘上的棋子变化	
预期结果： 棋盘上的棋子变成上一步棋局	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏是否支持悔棋功能（是/否）	
测试结果： 通过/不通过	

（3）棋子动画翻转功能的测试，主要内容如表 14.3 所示。

表 14.3　棋子动画翻转功能的测试

测试编号：1.7.3	类别：A
项　　目：黑白棋测试	
分 项 目：棋子动画翻转功能的测试	
测试目的：测试游戏中的棋子是否能够根据规则进行动画翻转	
测试配置：	
预置条件： 黑白棋游戏已经开始； 由玩家执黑方； 鼠标已经准备好	
测试步骤： 记录当前棋盘棋子放置状态； 在棋盘中的允许放置棋子位置按鼠标左键	
预期结果： 根据规则，横向、纵向和对角线上的棋子都变成玩家的黑子	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	

续表

测试记录： 游戏中棋子是否能够根据规则进行动画翻转（是/否）
测试结果： 通过/不通过

（4）游戏胜负判断功能的测试，主要内容如表 14.4 所示。

表 14.4　游戏胜负判断功能的测试

测试编号：1.7.4	类别：A
项　　目：黑白棋测试	
分 项 目：游戏胜负判断功能的测试	
测试目的：测试当一方获得胜利时，游戏能否给出正确的判断	
测试配置：	
预置条件： 游戏中已经有黑/白方差一步胜出	
测试步骤： 将黑方的棋子放入最后一步棋盘空格中； 或者将白方的棋子放入最后一步棋盘空格中	
预期结果： 游戏提示黑方胜出； 或者提示白方胜出	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 当一方获得胜利时，游戏能否给出正确的判断（是/否）	
测试结果： 通过/不通过	

（5）背景音乐播放功能的测试，主要内容如表 14.5 所示。

表 14.5　背景音乐播放功能的测试

测试编号：1.7.5	类别：A
项　　目：黑白棋测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试黑白棋游戏能否支持播放背景音乐	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	

续表

判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏能否支持播放背景音乐（是/否）
测试结果： 通过/不通过

（6）帮助功能的测试，主要内容如表 14.6 所示。

表 14.6　帮助功能的测试

测试编号：1.7.6	类别：A
项　　目：黑白棋测试	
分 项 目：帮助功能的测试	
测试目的：测试黑白棋游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选中“游戏帮助”|“帮助”菜单栏	
预期结果： 出现游戏帮助提示，说明游戏操作方法	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 黑白棋游戏是否有帮助提示功能（是/否）	
测试结果： 通过/不通过	

（7）人机对战功能的测试，主要内容如表 14.7 所示。

表 14.7　人机对战功能的测试

测试编号：1.7.7	类别：A
项　　目：黑白棋测试	
分 项 目：人机对战功能的测试	
测试目的：测试游戏中人机对战功能是否正确	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 先由玩家执黑先落一子； 查看电脑玩家能够在有效的地方落下白子	

续表

预期结果： 电脑玩家能够在有效的地方落下白子，并成功对棋盘上的棋子进行翻转
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏中人机对战功能是否正确（是/否）
测试结果： 通过/不通过

（8）人工智能等级设置功能的测试，主要内容如表 14.8 所示。

表 14.8 人工智能等级设置功能的测试

测试编号：1.7.8	类别：C
项　　目：黑白棋测试	
分 项 目：人工智能等级设置功能的测试	
测试目的：测试游戏中人工智能的等级是否可以设置	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“高”菜单栏； 选中“游戏设置”|“中”菜单栏； 选中“游戏设置”|“低”菜单栏； 开始游戏直到结束	
预期结果： 发现电脑玩家获得胜利的机会变多	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中的人工智能的等级是否可以设置（是/否）	
测试结果： 通过/不通过	

14.4 黑白棋游戏的界面实现

黑白棋游戏的 Visual C++工程采用 MFC 对话框模式进行开发。本节主要讲解黑白棋游戏各个功能模块的代码实现。

14.4.1 游戏菜单的实现

在黑白棋游戏中，通过如下几步即可实现游戏的菜单。

（1）在黑白棋游戏工程的资源中添加一个菜单资源，其属性如表 14.9 所示。

表 14.9　主菜单属性

ID	类　别	说　明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
IDR_EXIT_GAME	菜单栏	退出游戏
IDR_LEVEL_HIGH	选择菜单	游戏等级（高）
IDR_LEVEL_NOR	选择菜单	游戏等级（中）
IDR_LEVEL_LOW	选择菜单	游戏等级（低）
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于

（2）给每个菜单栏添加响应函数到 COthelloDlg 类中。
（3）菜单响应函数的实现，如代码 14.1 所示。

代码 14.1　菜单响应函数的实现

```
01  BEGIN_MESSAGE_MAP(COthelloDlg, CDialog)
02      ON_COMMAND(IDR_ABOUT, OnAbout)
03      ON_COMMAND(IDR_EXIT_GAME, OnExitGame)
04      ON_COMMAND(IDR_GAME_START, OnGameStart)
05      ON_COMMAND(IDR_HELP, OnHelp)
06      ON_COMMAND(IDR_LEVEL_HIGH, OnLevelHigh)
07      ON_COMMAND(IDR_LEVEL_LOW, OnLevelLow)
08      ON_COMMAND(IDR_LEVEL_NOR, OnLevelNor)
09      ON_COMMAND(IDR_PLAY_MUSIC, OnPlayMusic)
10  END_sMESSAGE_MAP()
11  ...                                        //省略部分代码，请查阅光盘源码
12  void COthelloDlg::OnAbout()
13  {
14      CAboutDlg dlg;                         //创建关于对话框类对象
15      dlg.DoModal();                         //弹出关于对话框
16  }
17
18  void COthelloDlg::OnExitGame()
19  {
20      CDialog::OnCancel();                   //调用基类退出函数
21  }
22
23  void COthelloDlg::OnGameStart()
24  {
25      GameStart();                           //调用游戏开始接口函数
26  }
27
28  void COthelloDlg::OnHelp()
29  {
30      CHelpDlg dlg;                          //创建帮助对话框类对象
31      dlg.DoModal();                         //弹出帮助对话框
32  }
33
34  void COthelloDlg::OnLevelHigh()
```

```
35  {
36      CWnd*   pMain   =   AfxGetMainWnd();//得到主窗口指针
37      CMenu*   pMenu   =   pMain->GetMenu();
38                                      //设置“高”等级被选中状态
39      pMenu->CheckMenuItem(IDR_LEVEL_HIGH,
40                      MF_BYCOMMAND | MF_CHECKED);
41      pMenu->CheckMenuItem(IDR_LEVEL_LOW,
42                      MF_BYCOMMAND | MF_UNCHECKED);
43      pMenu->CheckMenuItem(IDR_LEVEL_NOR,
44                      MF_BYCOMMAND | MF_UNCHECKED);
45      g_iGameLevel = LEVEL_HIGH;
46  }
47
48  void COthelloDlg::OnLevelLow()          //低等级菜单响应
49  {
50      CWnd*   pMain   =   AfxGetMainWnd();
51      CMenu*   pMenu   =   pMain->GetMenu();
52      pMenu->CheckMenuItem(IDR_LEVEL_HIGH,
53                      MF_BYCOMMAND | MF_UNCHECKED);
54                                      //设置“低”等级菜单栏被选中状态
55      pMenu->CheckMenuItem(IDR_LEVEL_LOW,
56                      MF_BYCOMMAND | MF_CHECKED);
57      pMenu->CheckMenuItem(IDR_LEVEL_NOR,
58                      MF_BYCOMMAND | MF_UNCHECKED);
59      g_iGameLevel = LEVEL_LOW;
60  }
61
62  void COthelloDlg::OnLevelNor()          //中等级菜单响应
63  {
64      CWnd*   pMain   =   AfxGetMainWnd();
65      CMenu*   pMenu   =   pMain->GetMenu();
66      pMenu->sCheckMenuItem(IDR_LEVEL_HIGH,
67              MF_BYCOMMAND | MF_UNCHECKED);
68      pMenu->CheckMenuItem(IDR_LEVEL_LOW,
69              MF_BYCOMMAND | MF_UNCHECKED);
70                                      //设置“中”等级菜单栏被选中状态
71      pMenu->CheckMenuItem(IDR_LEVEL_NOR,
72              MF_BYCOMMAND | MF_CHECKED);
73      g_iGameLevel = LEVEL_NOR;
74  }
75  void COthelloDlg::OnPlayMusic()         //播放音乐菜单响应
76  {
77      CWnd*   pMain   =   AfxGetMainWnd();
78      CMenu*   pMenu   =   pMain->GetMenu();
79      //判断播放音乐菜单当前状态
80      BOOL bCheck = (BOOL)pMenu->GetMenuState(IDR_PLAY_MUSIC,
        MF_CHECKED);
81
82      if(m_bStart)
83      {
84          if(bCheck)
85          {
86              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
87                          MF_BYCOMMAND | MF_UNCHECKED);
88          }
89          else
90          {
91              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
92                          MF_BYCOMMAND | MF_CHECKED);
```

```
93              }
94
95              PlayBackMusic(!bCheck);         //调用播放背景音乐功能函数
96          }
97  }
98
99  void COthelloDlg::InitMenu()
100 {
101                                             //初始化菜单
102     CWnd*   pMain   =   AfxGetMainWnd();
103     CMenu*   pMenu   =   pMain->GetMenu();
104     pMenu->CheckMenuItem(IDR_LEVEL_LOW, MF_BYCOMMAND | MF_CHECKED);
105     pMenu->CheckMenuItem(IDR_LEVEL_HIGH, MF_BYCOMMAND |
        MF_UNCHECKED);
106     pMenu->CheckMenuItem(IDR_LEVEL_NOR, MF_BYCOMMAND | MF_UNCHECKED);
107     pMenu->CheckMenuItem(IDR_PLAY_MUSIC,MF_BYCOMMAND| MF_UNCHECKED);
108 }
```

代码解析：代码第 99 行初始化菜单函数，是将各菜单进行初始状态设置。如果不做这一步，那么各菜单在初始时，全部是被选中状态。

14.4.2　游戏帮助对话框的实现

黑白棋游戏中的帮助是使用一个对话框来实现的。其实现步骤如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 14.5 所示。

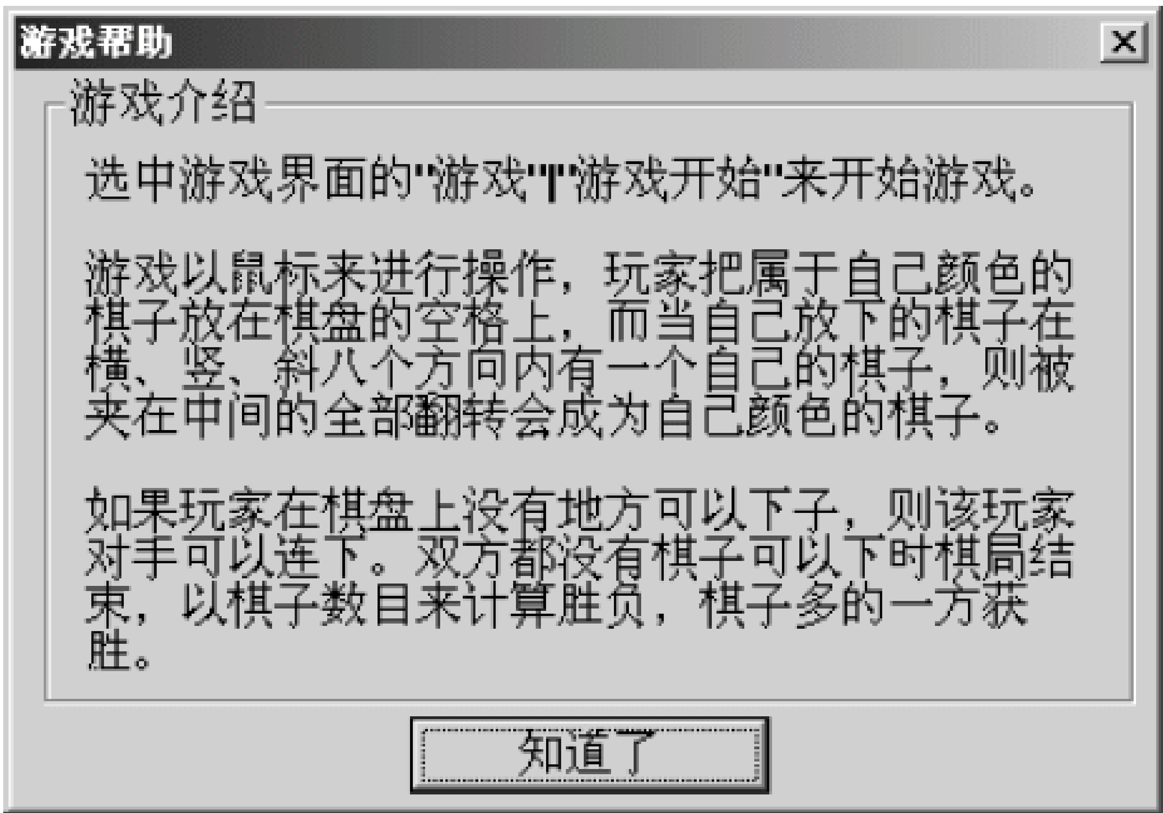


图 14.5　“游戏帮助”对话框

（2）编写一个 CHelpDlg 对话框类，主要是加载 IDD_HELP 对话框资源，通过资源中的文字说明对游戏操作方法进行描述。同时只包含单击“记得了”按钮的响应函数。其类声明如代码 14.2 所示。

代码 14.2　CHelpDlg 对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
04  // HelpDlg.h CHelpDlg 类声明头文件
05
06
07  /////////////////////////////////////////
08  // CHelpDlg 对话框类
09
10  class CHelpDlg : public CDialog             //公共继承于 CDialog 类
11  {
12  public:
13      CHelpDlg(CWnd* pParent = NULL);         //构造函数
14
15                                              //对话框资源
16      enum { IDD = IDD_HELP };                //加载资源
17
18                                              //重载函数
19      protected:
```

```
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                        //单击“确定”按钮响应函数声明
25      DECLARE_MESSAGE_MAP()
26  };
27
28  #endif
```

（3）CHelpDlg 对话框类的实现，需要实现对话框类的构造函数、析构函数和“知道了”按钮响应函数，如代码 14.3 所示。

代码 14.3　CHelpDlg 对话框类的实现

```
01  // HelpDlg.cpp CHelpDlg 类的实现源文件
02
03
04  #include "stdafx.h"                         //插入头文件
05  #include "Othello.h"
06  #include "HelpDlg.h"                        //插入类声明头文件
07
08  /////////////////////////////////////
09                                              //CHelpDlg 对话框类实现
10
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/)//构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);
19  }
20
21  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)
22  END_MESSAGE_MAP()
23
24  ///////////////////////////////////////////////
25  // CHelpDlg 消息响应函数
26
27  void CHelpDlg::OnOK()                       //单击“知道了”按钮响应函数
28  {
29      CDialog::OnOK();
30  }
```

14.4.3　游戏播放背景音乐的实现

播放游戏背景音乐，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。要实现这个功能，需要如下几个步骤。

（1）在工程文件中，添加 winmm.lib 静态库文件及头文件，参见第 5.4 节。

（2）实现 COthelloDlg 类中的 PlayBackMusic()成员函数，其代码如代码 14.4 所示。

代码 14.4　COthelloDlg 类的 PlayBackMusic 成员函数实现

```
01  #include <mmsystem.h>                    //插入系统 API 头文件
02  ...
03  void COthelloDlg::PlayBackMusic(BOOL bCheck)
04  {
05                                           //指定文件并播放
06      if(bCheck)
07      {                                    //播放指定音乐文件
08          sndPlaySound("music.wav",SND_ASYNC);
09      }
10      else
11      {                                    //停止播放
12          sndPlaySound(NULL,SND_PURGE);
13      }
14  }
```

代码解析：代码第 8 行是调用系统 API 函数播放指定音乐文件。

14.5　黑白棋游戏的核心算法设计与实现

在前面的章节中已经讲解了黑白棋游戏的菜单和各种对话框的实现。本节将对黑白棋游戏的核心算法的设计和实现进行讲解。

14.5.1　棋盘窗口类的设计

黑白棋的棋盘窗口类，主要负责显示游戏中的棋盘、棋子和棋子个数，同时还要管理绘图、输入及输出等内容。其主要有如下几个处理模块。

1．新游戏处理模块

开始新游戏处理模块，主要是把棋盘数据进行初始化，并让棋盘窗口进行重绘。

2．鼠标输入处理模块

利用截取窗口上鼠标输入的信息，得到当前玩家使用鼠标左键落子坐标。将坐标进行转换得到棋盘数组中的序号，把当前棋子颜色保存到该数组中。

3．延时处理模块

实现延时处理模块，需要通过如下几个步骤：

（1）得到当前系统的时钟计时。

（2）做一个消息循环，其是由 PeekMessage 和 PumpMessage 这个 API 函数组成。

（3）每循环一次，得到当前系统的时钟计时，并与延时相加得到超时时间。

（4）当超时时间到达时，退出循环，延时成功。

4．悔棋处理模块

实现悔棋处理模块，需要通过如下几个步骤：

（1）判断当前落子步数，如果小于 2，说明游戏还没有开始，不能悔棋。
（2）重新初始化当前棋盘数组，并把落子步骤数组的数据复制出来。
（3）根据当前落子步骤数组的数据，将棋盘的棋子位置进行恢复。

5．绘图处理模块

绘图处理模块主要通过遍历当前棋盘数组中的数据，根据每一行或者列的棋子类型的不同，进行相应的棋子绘图即可。

14.5.2　棋盘窗口类的实现

棋盘窗口类 CChessBoard 的实现，可以分为如下几个步骤：
（1）声明棋盘窗口类 CChessBoard，其中包含构造函数、析构函数、悔棋函数、翻转动画函数、绘图函数及鼠标输入处理函数等。其代码如代码 14.5 所示。

代码 14.5　棋盘窗口类 CChessBoard 的声明

```
01  #ifndef __CHESS_BOARD_H__
02  #define __CHESS_BOARD_H__
03  #include "DataStruct.h"                     //插入数据定义头文件
04
05  #define COL_WIDTH 45                        //定义列宽度
06  #define ROW_WIDTH 45                        //定义行宽度
07
08  class CChessBoard : public CWnd             //棋盘窗口类公有继承于窗口类
09  {
10  private:
11      CBitmap  m_bitBlackChess;               //定义黑子图片对象
12      CBitmap  m_bitWhiteChess;               //定义白子图片对象
13      CBitmap  m_bitChessBoard;               //定义棋盘图片对象
14      CBitmap  m_motive[8];                   //定义翻转动画图片对象
15      int      m_iMotiveNumber;               //当前动画序号
16      bool     m_bPlayMotive;                 //翻转标志
17      int      m_iMotivex, m_iMotivey;
18  public:
19      board_type m_oChessBoard;               //棋盘数组
20      CChessBoard();                          //构造函数
21
22  public:
23     void NewGame();                          //开始新游戏接口函数
24      void MoveBack();                        //悔棋接口函数
                                                //棋子翻转动画函数
25      void PlayMotive(int row, int col, UINT8 obcolor);
26
27  public:                                     //窗口创建函数
28      virtual BOOL Create(RECT &rect, CWnd * pParentWnd, UINT nID);
29
30  public:
31      virtual ~CChessBoard();                 //析构函数
32
33  protected:
34      afx_msg void OnPaint();                 //绘图函数
35      afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

```
36                                              //左键响应函数
37      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
38      afx_msg void OnComRun(WPARAM wParam, LPARAM lParam);
39      afx_msg void OnTranChess(WPARAM wParam, LPARAM lParam);
40      DECLARE_MESSAGE_MAP()
41  };
42
43  #endif
```

（2）实现棋盘窗口类 CChessBoard 的构造函数、析构函数、窗口建立函数，绘图函数及鼠标输入处理等函数，其代码如代码 14.6 所示。

代码 14.6　棋盘窗口类 CChessBoard 的基本函数实现

```
01  #include "stdafx.h"                         //插入头文件
02  #include "ChessBoard.h"
03  #include "resource.h"
04
05  UINT8 g_bStart = 0;                         //初始化开始状态为假
06  CChessBoard::CChessBoard()
07  {
08      m_iMotiveNumber=0;                      //初始化各成员变量
09      m_iMotivex = m_iMotivey=0;
10      m_bPlayMotive = FALSE;
11      init_board(&m_oChessBoard);
12  }
13
14  CChessBoard::~CChessBoard()
15  {
16  }
17                                              //消息映射函数表
18  BEGIN_MESSAGE_MAP(CChessBoard, CWnd)
19      ON_WM_PAINT()
20      ON_WM_LBUTTONDOWN()
21      ON_WM_CREATE()
22      ON_MESSAGE(UM_COMRUN, OnComRun)
23      ON_MESSAGE(WM_TRANCHESS, OnTranChess)
24  END_MESSAGE_MAP()
25  //窗口建立函数
26  BOOL CChessBoard::Create(RECT &rect, CWnd *pParentWnd, UINT nID)
27  {                                           //注册窗口类
28      CString szClassName = AfxRegisterWndClass(CS_CLASSDC|CS_SAVEBITS|
29                                CS_HREDRAW|CS_VREDRAW,
30                                0, (HBRUSH)CBrush(RGB(0,0,255)), 0);
31      rect.right = rect.left + 380+3;         //设置窗口右坐标，相当于指定大小
32      rect.bottom = rect.top +380+3;
33                                              //创建窗口
34      if(!CWnd::CreateEx(WS_EX_CLIENTEDGE, szClassName, _T(""),
35                         WS_CHILD|WS_VISIBLE|WS_TABSTOP, rect,
36                         pParentWnd, nID, NULL)) //WS_EX_CLIENTEDGE{
37              return FALSE;
38      }
39      UpdateWindow();                         //更新窗口
40      //加载各类图片
41      m_bitBlackChess.LoadBitmap(IDB_BLACKCHESS);
42      m_bitChessBoard.LoadBitmap(IDB_CHESSBOARD);
43      m_bitWhiteChess.LoadBitmap(IDB_WHITECHESS);
44      m_motive[0].LoadBitmap(IDB_WHITECHESS);
45      m_motive[1].LoadBitmap(IDB_TURN1); //加载转动图片 1
```

```
46       m_motive[2].LoadBitmap(IDB_TURN2); //加载转动图片 2
47       m_motive[3].LoadBitmap(IDB_TURN3); //加载转动图片 3
48       m_motive[4].LoadBitmap(IDB_TURN4); //加载转动图片 4
49       m_motive[5].LoadBitmap(IDB_TURN5); //加载转动图片 5
50       m_motive[6].LoadBitmap(IDB_TURN6); //加载转动图片 6
51       //加载黑棋
52       m_motive[7].LoadBitmap(IDB_BLACKCHESS);
53
54       return TRUE;
55   }
56   //窗口绘图函数
57   void CChessBoard::OnPaint()
58   {
59       CPaintDC dc(this);
60       CDC imgdc;
61       imgdc.CreateCompatibleDC(&dc);         //创建绘图 DC
62       imgdc.SelectObject(&m_bitChessBoard);//设置绘图对象
         //进行贴图处理
63      dc.BitBlt(0, 0, 380, 380, &imgdc,0,0,SRCCOPY);
64       if(m_bPlayMotive)                      //判断是否是动画
65       {
66           imgdc.SelectObject(&m_motive[m_iMotiveNumber]);
67           dc.BitBlt(m_iMotivex, m_iMotivey, 39, 39, &imgdc, 0, 0,
             SRCCOPY);
68           return;
69       }
70
71       for(int i=0; i<BOARD_ROWS; i++)
72       {
73           for(int j=0; j<BOARD_COLS; j++)
74           {
75               if(m_oChessBoard.board[i+1][j+1] == CHESS_BLACK)
76               {                       //设置绘图对象，并进行黑棋子的贴图处理
77                   imgdc.SelectObject(&m_bitBlackChess);
78                   dc.BitBlt(j*COL_WIDTH+24,
79                       i*ROW_WIDTH+24, 39, 39, &imgdc,0,0,SRCCOPY);
80               }
81               else if(m_oChessBoard.board[i+1][j+1] == CHESS_WHITE)
82               {                       //设置绘图对象，并进行白棋子的贴图处理
83                   imgdc.SelectObject(&m_bitWhiteChess);
84                   dc.BitBlt(j*COL_WIDTH+24,
85                       i*ROW_WIDTH+24, 39, 39, &imgdc,0,0,SRCCOPY);
86               }
87           }
88       }
89   }
90   //鼠标左键响应函数
91   void CChessBoard::OnLButtonDown(UINT nFlags, CPoint point)
92   {
93
94       BYTE row = (point.y-22)/ROW_WIDTH+1;//根据 Y 坐标得到行数
95       BYTE col = (point.x-22)/COL_WIDTH+1;//根据 X 坐标得到列数
96
97
         if(do_move_chess(&m_oChessBoard,row*10+col,~computer_
         side&3,m_hWnd))
98       {
99           UINT16 wscore, bscore;
```

```
100         get_chess_score(&m_oChessBoard, wscore, bscore);
101          GetParent()->SendMessage(UM_RECALC,WPARAM(wscore),
             LPARAM(bscore));
102          PostMessage(UM_COMRUN);
103      }
104      else
105      {
106          MessageBeep(MB_OK);
107      }
108      CWnd::OnLButtonDown(nFlags, point);
109 }
110
111 int CChessBoard::OnCreate(LPCREATESTRUCT lpCreateStruct)
112 {
113      if (CWnd::OnCreate(lpCreateStruct) == -1)
114          return -1;
115
116      EndWaitCursor();                        //窗口创建完毕时，自动终止等待光标
117      return 0;
118 }
```

代码解析：代码第 71～88 行是利用循环语句对棋盘数组进行遍历，当读取到的数据是白棋时，就将白棋图片加载并显示出来。对于黑棋也是同样的方法。

（3）实现 CChessBoard 类的扩展功能函数，包括延时函数、动画翻转函数、悔棋函数、新游戏接口函数、变更棋子函数及电脑下棋接口函数。其代码如代码 14.7 所示。

代码 14.7　CChessBoard 类的扩展功能函数实现

```
01  //延时函数
02  void delay(INT32 millisecond)
03  {
04      clock_t start = clock();
05      do
06      {                                        //消息循环
07          MSG msg;
08          if (::PeekMessage( &msg, NULL, 0, 0, PM_NOREMOVE ) )
09          {
10              if ( !AfxGetApp()->PumpMessage())
11              {
12              ::PostQuitMessage(0);            //发送退出消息，退出程序
13               return;
14              }
15          }
16      }while(clock()<start+millisecond);//比较超时没有
17  }
18  //悔棋函数
19  void CChessBoard::MoveBack()
20  {
21      if(cur_step<2)
22      {                                        //如果当前步骤小 2，说明没有开始游戏
23          return;
24      }
25     UINT8 comside = computer_side;            //得到当前电脑下的棋子信息
26     UINT8 step = cur_step;                    //保存当前步骤
27     INT16 movearray[64];
28                                         //把下棋步骤数组中的数据复制到移动数组中
29     memcpy(movearray, step_array, 64*sizeof(INT16));
30     init_board(&m_oChessBoard);
```

```
31    computer_side = comside;
32    UINT8 col= CHESS_BLACK;
33    for(int i=0; i<step-2; i++, col = ~col & 3)
34    {
35        do_move_chess(&m_oChessBoard, movearray[i], col, 0);
36    }
37    OnPaint();
38    Invalidate();
39  }
40  //改变棋子接口函数
41  void CChessBoard::OnTranChess(WPARAM wParam, LPARAM lParam)
42  {
43      int row = wParam/10-1;
44      int col = wParam%10-1;
45     CRect r(col*COL_WIDTH+22, row*ROW_WIDTH+22,
46             col*COL_WIDTH+COL_WIDTH+22,
47              row*ROW_WIDTH+ROW_WIDTH+22);
48
49      m_bPlayMotive = FALSE;      //动画标志为假
50      OnPaint();                  //调用绘图函数
51      InvalidateRect(&r);         //使指定范围失效，可以进行重绘
52
53     if((lParam>>16) !=0)
54      PlayMotive(row, col, UINT8(lParam));
55  }
56  //由电脑下棋转换
57  void CChessBoard::OnComRun(WPARAM wParam, LPARAM lParam)
58  {
59                                  //调用由人工智能处理接口函数
60      computer_play(&m_oChessBoard, m_hWnd);
61     UINT16 wscore, bscore;
62                                  //得到当前棋盘上的棋子个数
63      get_chess_score(&m_oChessBoard, wscore, bscore);
64      GetParent()->SendMessage(UM_RECALC,
65                     WPARAM(wscore|0x80000000), LPARAM(bscore));
66  }
67  //新游戏
68  void CChessBoard::NewGame()
69  {
70      if(cur_step >0)
71      {
72          if(MessageBox("开始新游戏吗?", "黑白棋",
73              MB_YESNO|MB_ICONQUESTION) == IDYES)
74          {
75             g_bStart = 1;
76             init_board(&m_oChessBoard);
77             Invalidate();
78          }
79      }
80  }
81  //播放棋子翻动动画
82  void CChessBoard::PlayMotive(int row, int col, UINT8 obcolor)
83  {
84      m_iMotivex = col*COL_WIDTH+24;
85      m_iMotivey = row*COL_WIDTH+24;
86      CRect r(m_iMotivex, m_iMotivey,
87          m_iMotivex+COL_WIDTH,
88          m_iMotivey +ROW_WIDTH);
89      m_bPlayMotive = TRUE;
```

```
90        if(obcolor == CHESS_BLACK)
91        {//把棋子从白面向黑面翻转
92            for(m_iMotiveNumber =0; m_iMotiveNumber<8; m_iMotiveNumber++)
93            {
94                OnPaint();
95                InvalidateRect(&r);
96                delay(50);
97            }
98        }
99        else
00        {//把棋子从黑面向白面翻转
101           for(m_iMotiveNumber =7; m_iMotiveNumber>=0; m_iMotiveNumber--)
102           {
103               OnPaint();
104               InvalidateRect(&r);
105               delay(50);
106           }
107       }
108       m_bPlayMotive = FALSE;
109 }
```

代码解析：代码第 19 行的悔棋函数是利用将整个下棋的步骤先存放到移动棋子数组中，然后再遍历除最后两步外的这个移动棋子数组数据，将前面所有下过的棋子重新下一次，从而整个棋盘实现悔棋。但要注意，这里只是对棋子数组进行操作，时间上非常快，所以整个过程不会显示出来。

技巧： 游戏中棋子动画翻转主要是依靠不断地重新在同一个位置绘制指定的图片，来产生动画效果。从黑到白或者从白到黑，只是遍历数组时，序号从前向后，或者从后向前的不同而已。

14.5.3　人工智能模块的设计

人工智能模块，主要的设计思想是从棋盘的当前状态构造博弈树，到达终局状态时计算状态分，并回归到当前状态，以求出最有利的一步（部分回溯值最大的一个子结点）。其要点如下所述。

1．终局状态

（1）棋局已结束。

（2）当前颜色已无处可下子。

（3）博弈树的深度已到指定的深度。

2．博弈树的构造方法

采用深度优先的方法节省构造过程中的内存使用，构造过程使用堆栈空间存储子结点，已完成构造的分枝可抛弃，以达到节省内存（使用递归程序）。算法简单描述如下几个步骤：

（1）如果当前棋局为终局状态，则返回状态分。

（2）从当前棋局的状态出发，找出一个可走的步数，试走此步。新状态扩展为当前棋局的一个子结点。此子结点做为新的当前状态递归调用。

3. 构造过程的α-β裁减

（1）α 裁减

在考虑轮到棋手下棋的一个亲结点及轮到对手下棋的一个子结点时，如果该子结点的数值已经小于或等于其亲结点的回溯值，那么就不需要对该结点或者其后续结点做更多的处理了。计算的过程可以直接返回到亲结点上。

（2）β 裁减

在考虑轮到对手下棋的一个亲结点及轮到棋手下棋的一个子结点时，如果该子结点的部分回溯值已经大于或等于其亲结点的部分回溯值，那么就不需要对该子结点或者其后裔结点做更多的处理了。计算过程可以直接返回到亲结点上。

4. 棋局的状态估值函数

此函数量化的方法描述棋局某一状态下某一方棋子的形式。对棋局形式的正确分析直接关系到电脑棋力的高低考虑黑白棋的规则，棋局结束时，棋盘上哪一方的棋子多则哪一方获胜，比较好的方法是计算棋盘上所有不可能被对方吃掉的棋子作为状态分。

棋盘上任一个棋子是否能被对方吃掉由其 4 个方向其他棋格的状态决定，这 4 个方向分别是：

- 左上到右下的斜线。
- 右上到左下的斜线。
- 水平方向。
- 垂直方向。

一个棋子如果在这 4 个方向上都受保护，则认为此棋子不可能被对方吃掉。

5. 判断棋子是否受保护

一个棋子是否在某一个方向上受保护，可用下面的方法来判断。当出现下面的两种情况之一时，可以认为此棋子在此方向上受保护：

（1）此方向上与此棋子相联的一方棋子已到达边界。

（2）此方向上与此棋子相联的一方棋子两边没有空格子（对方棋子，或边界）。

由此可知，棋盘空时，四个角上的棋子一定是不可被吃的棋子。

6. 状态分的计分方法

- 不可被吃掉的棋子计四分。
- 三个方向上受保护的棋子计三分。
- 两个方向上受保护的棋子计二分。
- 一个方向上受保护的棋子计一分。

另外于角的重要性，下面的情况出现时被视为危险状态将会被减分。例如，角边上有一方棋子，但在到达角的方向上没有受到保护则减分。

7. 棋盘的数据结构

每一个棋格用一个字节记，低两位记录棋子颜色，高四位记录此子在四个方向上是否

受保护的情况。如果一个棋格为空，则对应字节为 0X00，为避免边界检查，在棋盘四周加一个边框，边框对应值为 0XFF。

14.5.4　人工智能模块的实现

人工智能模块的实现，分为如下几个步骤。

（1）实现人工智能模块中比较简单的功能函数。包含初始化游戏函数，游戏结束函数、调用电脑下棋接口函数及得到当前各颜色棋子个数接口函数。其代码如代码 14.8 所示。

代码 14.8　人工智能模块中的基本功能函数实现

```
01  /*初始化棋盘数组*/
02  void init_board(board_type *board_ptr)
03  {
04      memset(board_ptr, 0, sizeof(board_type));
05      /*初始化棋盘数据 */
06      memset(board_ptr->board[0], 0xff, 10);
07      memset(board_ptr->board[9], 0xff, 10);
08      for(int i=0; i<9; i++)
09      {
10          board_ptr->board[i][0] = board_ptr->board[i][9] =0xff;
11      }
12
13      /*初始化棋子数组*/
14      board_ptr->board[4][4] = board_ptr->board[5][5] = CHESS_WHITE;
15      board_ptr->board[4][5] = board_ptr->board[5][4] = CHESS_BLACK;
16      cur_step = 0;
17      computer_side = CHESS_WHITE;         //设置电脑方为白方
18  }
19
20  /*获得当前不同颜色棋子的个数*/
21  void get_chess_score(board_type *board_ptr, UINT16 &iWscore, UINT16
    &iBscore)
22  {
23      iWscore =0; iBscore =0;
24      for(INT16 i=1; i<=BOARD_ROWS; i++)
25          for(INT16 j=1; j<=BOARD_COLS; j++)
26          {
27              if(board_ptr->board[i][j]  == CHESS_BLACK)
28                  iBscore++;                //黑棋子增加
29              else  if(board_ptr->board[i][j] == CHESS_WHITE)
30                  iWscore++;                //白棋子增加
31          }
32  }
33  /*游戏结束处理函数*/
34  void game_over(board_type *board_ptr, HWND hwnd)
35  {
36      UINT16 wscore, bscore;
37      char strcomwin[]="虽然你很厉害,但我还是赢了你!";
38      char struserwin[]="让你一次,下次你可没这么走运了!";
39      char strdogfall[]="我没好好下,你才有机会平局!";
40      char *text;
41      get_chess_score(board_ptr, wscore, bscore);
42      g_bStart = 0;
43      if(computer_side == CHESS_WHITE)
```

```
44      {/*根据当前双方的棋子数得到结果*/
45          if(wscore > bscore)
46          {
47              text = strcomwin;          //得到显示字符
48          }
49          else if(wscore <bscore)
50          {
51              text = struserwin;         //得到显示字符
52          }
53          else
54          {
55              text = strdogfall;         //得到显示字符
56          }
57      }
58      else
59      {
60          if(wscore > bscore)
61              text = struserwin;         //得到显示字符
62          else if(wscore <bscore)
63              text = strcomwin;          //得到显示字符
64          else text = strdogfall;
65      }
66      MessageBox(hwnd,  text, "黑白棋", MB_OK|MB_ICONINFORMATION);
67  }
68  /*调用电脑下棋接口函数*/
69  void computer_play(board_type *board_ptr, HWND hwnd)
70  {
71      cur_depth =0;
72      tree_node_type node;                   //创建当前结点树
73      INT16 affected_list[MAX_AFFECTED_PIECES];
74  start:
75      memcpy(&node.board, board_ptr, sizeof(board_type));
76      node.movepos =0;
77      if(cur_step>= STEP_MONMENT2)
78      {
79          extend_node_two(&node, NULL, computer_side);
80      }
81      else if(cur_step > STEP_MONMENT1)
82      {
83          max_depth = depth2[g_iGameLevel];//根据难度进行
84          extend_node_one(&node, NULL, computer_side);
85      }
86      else
87      {
88          max_depth = depth1[g_iGameLevel];
89          extend_node_one(&node, NULL, computer_side);
90      }
91      /*执行移动棋子操作*/
92      if(!do_move_chess(board_ptr, node.movepos, computer_side, hwnd))
93      {
94          if(!find_move(board_ptr, 11, (~computer_side)&0x03,
            affected_list))
95          {
96              game_over(board_ptr, hwnd); //游戏结束
97              return;
98          }
99          else
100         {
101             MessageBox(hwnd,"我没棋下了,你再走一步!",
```

```
102                 "黑白棋", MB_OK|MB_ICONINFORMATION);
103             return;
104         }
105     }
106     else
107     {   /*查找是否有可以移动的棋子落下*/
108         if(!find_move(board_ptr, 11, (~computer_side)&0x03,
            affected_list))
109         {
110             if(!find_move(board_ptr, 11, computer_side,
                affected_list))
111             {
112                 game_over(board_ptr, hwnd);
113                 return;
114             }
115             else
116             {
117
118                 MessageBox(hwnd,"你没棋下了,
119                     我再走一步!", "黑白棋", MB_OK|MB_ICONINFORMATION);
120
121                 goto start;
122             }
123         }
124     }
125 }
```

代码解析：代码第 69 行的电脑下棋函数 computer_play()，就是根据要点中的 2～6 进行的代码实现。

（2）实现在当前棋盘中，查找直线受保护的棋子功能函数。其中包括水平方向上的棋子、垂直方向上的棋子等，如代码 14.9 所示。

代码 14.9　查找直线受保护棋子功能函数实现

```
01 /*找出所有在水平方向受保护的 obcolor 方的棋子，并累计分数*/
02 INT16 scan_horiz_aixes(board_type *board_ptr, UINT8 obcolor)
03 {
04     /*扫描 8 个水平方向*/
05     INT16 score=0;                      //初始化各变量
06     UINT8 *cur_ptr, *stop_ptr;
07     UINT8  piece[4][2];
08     UINT8 count=0, tmpscore;
09     UINT8 bFull;
10     for(UINT8 row=1; row<9; row++){     //遍历 8 行数据
11        tmpscore = (row == 1 || row == 8) ? 10:2;
12         cur_ptr = &board_ptr->board[row][1];
13         stop_ptr= &board_ptr->board[row][9];
14         bFull = TRUE;
15         count=0;
16         while(cur_ptr < stop_ptr)       //判断当前和停止棋子
17         {
18             if(*cur_ptr == obcolor){
19                 piece[count][0]  = cur_ptr - &board_ptr->board[row][0];
20                 while(*cur_ptr == obcolor)
21                     cur_ptr ++;
22                 piece[count++][1] = cur_ptr - &board_ptr->board
                   [row][0];
23             }
```

```
24              if(!*cur_ptr)
25                  bFull = FALSE;
26              cur_ptr++;
27          }
28          while(count--){
29              UINT8 nums = (piece[count][1]-piece[count][0]);
30              if(bFull || piece[count][0]==1 || piece[count][1] == 9)
31                  score += nums;
32              if(piece[count][0]==1 || piece[count][1] == 9)
33                  score += tmpscore;
34              else if(!bFull && (piece[count][0] == 2 || piece[count][1]
                == 8)
35                      && (row == 1 || row == 8))
36                  score -= tmpscore;
37          }
38      }
39      return score;                       //返回状态分
40  }
41  /*找出所有在垂直方向受保护的 obcolor 方的棋子，并累计分数*/
42  INT16 scan_vertical_aixes(board_type *board_ptr, UINT8 obcolor)
43  {
44      INT16 score=0;
45      UINT8 *cur_ptr, *stop_ptr;
46      UINT8  piece[4][2];
47      UINT8 count=0, tmpscore;
48      UINT8 bFull;
49      for(UINT8 col=1; col<9; col++)
50      {
51         tmpscore = (col == 1 || col == 8) ? 10:2;
52          cur_ptr = &board_ptr->board[1][col];
53          stop_ptr= &board_ptr->board[9][col];
54          bFull = TRUE;
55          count=0;
56          while(cur_ptr < stop_ptr)       //判断当前和停止棋子
57          {
58              if(*cur_ptr == obcolor)
59              {
60                  piece[count][0]  = (cur_ptr - &board_ptr->board[0]
                    [col])/10;
61                  while(*cur_ptr == obcolor)
62                      cur_ptr += 10;
63                  piece[count++][1] = (cur_ptr - &board_ptr->board[0]
                    [col])/10;
64              }
65              if(!*cur_ptr)
66                  bFull = FALSE;
67              cur_ptr += 10;
68          }
69          while(count--)
70          {
71              UINT8 nums = (piece[count][1]-piece[count][0]);
72              if(bFull || piece[count][0]==1 || piece[count][1] == 9)
73                  score += nums;
74              if(piece[count][0]==1 || piece[count][1] == 9)
75                  score += tmpscore;
76              else if(!bFull && (piece[count][0] == 2 || piece[count][1]
                == 8)
77                  && (col == 1 || col == 8))
78                  score -= (tmpscore<<1);
79          }
```

```
80      }
81          return score;                    //返回状态分
82  }
```

代码解析：根据前面要点 5 的说明，代码第 2 行的 scan_horiz_aixes()函数就是实现查找所有在水平方向受保护的指定方的棋子，并累计分数。代码第 42 行 scan_vertical_aixes()函数是实现垂直方向的受保护的指定方的棋子。

（3）实现查找斜线方向上受保护的棋子的功能函数，包含右上到左下、左上到右下方向两个功能函数。其代码如代码 14.10 所示。

代码 14.10　查找斜线方向受保护棋子功能函数实现

```
01  /*找出所有在右上到左下方向受保护的 obcolor 方的棋子，并累计分数*/
02  INT16 scan_fd_aixes(board_type *board_ptr, UINT8 obcolor)
03  {
04      INT16 score =0;
05     UINT8 *cur_ptr, *stop_ptr, *base_ptr;
06      UINT8  piece[4][2];
07      UINT8 count=0, tmpscore;
08      UINT8 bFull;
09      for(INT8 aixes = -5; aixes <= 5; aixes++){
10         tmpscore = (aixes == 0) ? 10:2;
11          if(aixes <=0){
12              base_ptr = cur_ptr = &board_ptr->board[1][8+aixes];
13              stop_ptr = &board_ptr->board[9+aixes][0];
14          }else   {
15              base_ptr = cur_ptr = &board_ptr->board[aixes+1][8];
16              stop_ptr= &board_ptr->board[9][aixes];
17          }
18          bFull = TRUE;
19          count=0;
20          while(cur_ptr < stop_ptr){          //判断当前和停止棋子
21              if(*cur_ptr == obcolor){
22                  piece[count][0]  = cur_ptr - board_ptr->board[0];
23                  while(*cur_ptr == obcolor)
24                      cur_ptr += 9;
25                  piece[count++][1] = cur_ptr- board_ptr->board[0];
26              }
27              if(!*cur_ptr)
28                  bFull = FALSE;
29              cur_ptr += 9;
30          }
31          while(count--){
32              UINT8 nums = (piece[count][1]-piece[count][0])/9;
33              BOOL  toborder = (piece[count][0] == base_ptr -
                    board_ptr->board[0] ||
34                  piece[count][1] == stop_ptr - board_ptr->board[0]);
35              if(bFull || toborder)
36                  score += nums;
37              if((aixes == 1 || aixes == -1) && toborder)
38                  score -= tmpscore;
39              /*如果是这块棋到达边界*/
40              else if(toborder)
41                  score += tmpscore;
42              /*如果有棋在角边上，则扣分*/
43              else if(!bFull && (piece[count][0] == 27 ||
44                  piece[count][1] == 81))
45                  score -= (tmpscore<<1);
```

```
46              }
47          }
48          /*如果角边有棋子，则扣分*/
49          if(board_ptr->board[1][1] == obcolor)
50              score += 10;
51          else{
52              if(board_ptr->board[1][2]  == obcolor)
53                  score -=2;
54              if(board_ptr->board[2][1]  == obcolor)
55                  score -=2;
56              if(board_ptr->board[2][2]== obcolor)
57                  score -=2;
58          }
59          if(board_ptr->board[8][8] == obcolor)
60              score +=10;
61          else{
62              if(board_ptr->board[7][8]   == obcolor)
63                  score -=2;
64              if(board_ptr->board[8][7]== obcolor)
65                  score -=2;
66              if(board_ptr->board[7][7]== obcolor)
67                  score -= 2;
68          }
69         return score;                      //返回状态分
70     }
71     /*找出所有在左上到右下方向受保护的 obcolor 方的棋子，并累计分数*/
72     INT16 scan_bd_aixes(board_type *board_ptr,  UINT8 obcolor)
73     {
74         INT16 score =0;
75         UINT8 *cur_ptr, *stop_ptr, *base_ptr;
76          UINT8  piece[4][2];
77          UINT8 count=0, tmpscore;
78          UINT8 bFull;
79          for(INT8 aixes = -5; aixes <= 5; aixes++){
80             tmpscore = (aixes == 0) ? 10:2;
81              if(aixes <=0){
82                  base_ptr = cur_ptr = &board_ptr->board[1-aixes][1];
83                  stop_ptr = &board_ptr->board[9][9+aixes];
84              }else   {
85                  base_ptr = cur_ptr = &board_ptr->board[1][aixes+1];
86                  stop_ptr= &board_ptr->board[9-aixes][9];
87              }
88              bFull = TRUE;
89              count=0;
90              while(cur_ptr < stop_ptr){
91                  if(*cur_ptr == obcolor){
92                      piece[count][0]  = cur_ptr - board_ptr->board[0];
93                      while(*cur_ptr == obcolor)
94                          cur_ptr += 11;
95                      piece[count++][1]  = cur_ptr- board_ptr->board[0];
96                  }
97                  if(!*cur_ptr)
98                      bFull = FALSE;
99                  cur_ptr += 11;
100             }
101             while(count--){
102                 UINT8 nums = (piece[count][1]-piece[count][0])/11;
103                 BOOL  toborder = (piece[count][0] == base_ptr - board_ptr->
                    board[0]  ||
104                     piece[count][1] == stop_ptr - board_ptr->board[0]);
```

```
105             if(bFull || toborder)
106                 score += nums;
107             /*如果角边有棋子，则扣分*/
108             if((aixes == 1 || aixes == -1) && toborder)
109                 score -= tmpscore;
110             /*如果是这块棋到达边界*/
111             else if(toborder)
112                 score += tmpscore;
113             /*如果有棋在角边上，则扣分，主对角线方向*/
114             else if(!bFull && (piece[count][0] == 22 ||
115                 piece[count][1] == 88))
116                 score -= (tmpscore<<1);
117         }
118     }
119     /*如果角边有棋子，则扣分*/
120     if(board_ptr->board[1][8] == obcolor)
121         score += 10;
122     else{
123         if(board_ptr->board[1][7] == obcolor)
124             score -=2;
125         if(board_ptr->board[2][8] == obcolor)
126             score -=2;
127         if(board_ptr->board[2][7]== obcolor)
128             score -=2;
129     }
130     if(board_ptr->board[8][1] == obcolor)
131         score +=10;
132     else{
133         if(board_ptr->board[7][1]  == obcolor)
134             score -=2;
135         if(board_ptr->board[8][2]== obcolor)
136             score -=2;
137         if(board_ptr->board[7][2]== obcolor)
138             score -= 2;
139     }
140    return score;
141 }
```

代码解析：根据前面要点 5 的说明，代码第 2 行的 scan_fd_aixes()函数就是实现查找所有右上到左下方向受保护的指定方的棋子，并累计分数。代码第 72 行的 scan_bd_aixes()函数是实现左上到右下方向的受保护的指定方的棋子。

（4）实现人工智能模块中的计算当前棋盘状态函数、计算当前棋盘状态得分函数、移动棋子函数。其代码如代码 14.11 所示。

代码 14.11　人工智能模块中的计算和移动函数

```
01  /*计算当前棋盘状态*/
02  INT16 sample_calc_board_status(board_type *board_ptr, UINT8 obcolor)
03  {
04      INT16 score=0;
05      UINT8 *ptr = &board_ptr->board[1][1];
06      UINT8 *stop = &board_ptr->board[8][9];
07      UINT8 tmpcol = ~obcolor &0x03;
08      while(ptr<stop)
09      {
10          if(*ptr == obcolor)
11              score++;
12          else if(*ptr == tmpcol)
```

```
13              score--;
14          ptr++;
15      }
16      return score;
17  }
18
19  /*计算棋局 board_ptr 的状态分*/
20  INT16 calc_board_status(board_type *board_ptr, UINT8 obcolor)
21  {
22      INT16 score=0;
23      score += scan_horiz_aixes(board_ptr,  obcolor); //得到水平保护分数
24      score += scan_vertical_aixes(board_ptr, obcolor);//得到垂直保护分数
25      score += scan_bd_aixes(board_ptr, obcolor);    //得到斜向保护分数
26      score += scan_fd_aixes(board_ptr, obcolor);
27      UINT8 tmpcol = ~obcolor & 0x03 ;
28      if(board_ptr->board[1][1] == tmpcol)
29          score -= 44;
30      if(board_ptr->board[8][8] == tmpcol)
31          score -= 44;
32      if(board_ptr->board[1][8] == tmpcol)
33          score -= 44;
34      if(board_ptr->board[8][1] == tmpcol)
35          score -= 44;
36      return score;
37  }
38  /*移动棋子函数*/
39  UINT8 do_move_chess(board_type *board_ptr, UINT16 movepos,
40                      UINT8 obcolor, HWND hwnd)
41  {
42      INT16 affected_list[MAX_AFFECTED_PIECES];
43      INT16 num = find_move(board_ptr, movepos, obcolor, affected_list);
44      if(!num || affected_list[0] != movepos)
45          return 0;
46      for(int i=0; i<num; i++)
47      {
48          board_ptr->board[0][affected_list[i]] = obcolor;
49          if(hwnd)
50              ::SendMessage(hwnd, WM_TRANCHESS,
51                  WPARAM(affected_list[i]),LPARAM(i<<16|obcolor));
52      }
53      step_array[cur_step++] = movepos;
54      return 1;
55  }
```

代码解析：代码第 43 行是调用查找移动棋子位置函数，实现移动棋子的功能。当找到可以下子的位置后，就发送 WM_TRANCHESS 消息给主窗口，实现落子操作。

（5）实现人工智能模块中的核心功能函数。计算并寻找棋盘中可以下子的位置点，并根据当前游戏等级不同，搜索不同的深度。其主要是一个递归函数实现，其代码如代码 14.12 所示。

代码 14.12　核心功能函数实现

```
01  /*从 start_pos 出发找到一个可下子的点，返回受影响的子的个数，
02  affected_list 存放受影响的棋格的指针，第一个指针为落子的点*/
03  const INT16 delta_array[8] = {-11, 11, -9, 9, -1, 1, -10, 10};
04  INT16 find_move(board_type *board_ptr, INT16 start_pos,
05                  UINT8 obcolor, INT16 *affected_list)
```

```
06  {
07      UINT8 *cel_ptr = board_ptr->board[0] + start_pos;
08      UINT8 *stop_ptr = &board_ptr->board[8][9], *p;
09      INT16 *aff_ptr = affected_list+1, *hold_aff;
10      UINT8 aixes;
11      UINT8 thithercolor = THITHER_COLOR(obcolor);
12      while(1)
13      {
14          /*找到一个空格子*/
15          while(*cel_ptr)
16              if(++cel_ptr>=stop_ptr)
17                  return 0;
18              /*检查在 8 个方向上是否能吃掉对方的棋子，并记录被吃掉棋子的下标*/
19              for(aixes =0;aixes<8; aixes++)
20              {
21                  hold_aff = aff_ptr;
22                  p = cel_ptr + delta_array[aixes];
23                  while(*p == thithercolor)
24                  {
25                      *aff_ptr++ = p - board_ptr->board[0];
26                      p+= delta_array[aixes];
27                  }
28                  if(*p != obcolor)
29                      aff_ptr = hold_aff;
30              }
31              /*如果 cel_ptr 对应的点可以吃掉对方的子*/
32              if(aff_ptr - affected_list > 1)
33              {
34                  *affected_list = cel_ptr - board_ptr->board[0];
35                  return (aff_ptr - affected_list);
36              }
37              cel_ptr++;
38      }
39  }
40  /*从棋盘的一个状态出发，扩展此结点，并返回此结点的部分回溯值*/
41  void extend_node_one(tree_node_type *node_ptr,
42      tree_node_type *parent_ptr,UINT8 obcolor)
43  {
44      tree_node_type childnode;
45      INT16 affected_list[MAX_AFFECTED_PIECES];
46      INT16 start_pos = 11, num;
47      num = find_move(&node_ptr->board, start_pos, obcolor,
        affected_list);
48      /*如果是终局状态，则返回状态估值函数的值*/
49      if(++cur_depth == max_depth || num==0 )
50      {
51          /*如果一方 PASS 但没到棋局结束，要扣分*/
52          node_ptr->value = calc_board_status(&node_ptr->board,
            computer_side);
53          if(!num)
54          {
55              /*如果双方都没棋下*/
56              if(!find_move(&node_ptr->board, 11, ~obcolor&0x03,
                affected_list))
57                  return;
58
59              if(obcolor == computer_side)
60              {
61                  node_ptr->value -= 15;
```

```
62                  return ;
63              }
64              node_ptr->value += 15;
65          }
66          return;
67      }
68      /*初始化回溯值*/
69      node_ptr->value = (obcolor == computer_side)? -INITIAL_VALUE :
        INITIAL_VALUE;
70      memcpy(&childnode.board, &node_ptr->board, sizeof(board_type));
71      while(num)
72      {
73          while(num--)
74              childnode.board.board[0][affected_list[num]] = obcolor;
75          /*递归计算部分回溯值*/
76          UINT8 depth = cur_depth;
77          extend_node_one(&childnode, node_ptr, (~obcolor)&0x03);
78          cur_depth = depth;
79          /*如果此结点是棋手一方，则部分回溯值是子结点中最大的一个*/
80          if(obcolor == computer_side)
81          {
82              if(childnode.value > node_ptr->value)
83              {
84                  node_ptr->value = childnode.value;
85                  node_ptr->movepos = affected_list[0];
86              }
87          }
88          /*如果是对手一方，部分回溯值是子结点中最小的一个*/
89          else
90          {
91              if(childnode.value < node_ptr->value)
92              {
93                  node_ptr->value = childnode.value;
94                  node_ptr->movepos = affected_list[0];
95              }
96          }
97          /* α-β裁减的判断*/
98          if(parent_ptr)
99          {
100             if(obcolor != computer_side)
101             {
102                 /*α裁减*/
103                 if(node_ptr->value <= parent_ptr->value)
104                     return;
105             }
106             else
107             {
108                 /*β裁减*/
109                 if(node_ptr->value >= parent_ptr->value)
110                     return;
111             }
112         }
113         /*找到下一个可落子的点*/
114         start_pos = affected_list[0]+1;
115         memcpy(&childnode.board, &node_ptr->board, sizeof
            (board_type));
116         num = find_move(&childnode.board, start_pos, obcolor,
            affected_list);
117     }
```

```
118     return;
119 }
120 void extend_node_two(tree_node_type *node_ptr,
121     tree_node_type *parent_ptr,UINT8 obcolor)
122 {
123     tree_node_type childnode;
124     INT16 affected_list[MAX_AFFECTED_PIECES];
125     INT16 start_pos = 11, num;
126     num = find_move(&node_ptr->board, start_pos, obcolor,
        affected_list);
127     /*如果是终局状态，则返回状态估值函数的值*/
128     if(!num)
129     {
130         /*如果一方 PASS 但没到棋局结束，要扣分*/
131         node_ptr->value = sample_calc_board_status(&node_ptr->board,
            computer_side);
132         /*如果双方都没棋下*/
133         if(!find_move(&node_ptr->board, 11, ~obcolor&0x03,
            affected_list))
134             return;
135
136        if(obcolor == computer_side)
137         {
138             node_ptr->value -= 10;
139             return;
140         }
141         node_ptr->value += 10;
142         return;
143     }
144     /*初始化回溯值*/
145     node_ptr->value = (obcolor == computer_side)? -INITIAL_VALUE :
        INITIAL_VALUE;
146     memcpy(&childnode.board, &node_ptr->board, sizeof(board_type));
147     while(num)
148     {
149         while(num--)
150             childnode.board.board[0][affected_list[num]] = obcolor;
151         /*递归计算部分回溯值*/
152         UINT8 depth = cur_depth;
153         extend_node_two(&childnode, node_ptr, (~obcolor)&0x03);
154         cur_depth = depth;
155         /*如果此结点是棋手一方，则部分回溯值是子结点中最大的一个*/
156         if(obcolor == computer_side)
157         {
158             if(childnode.value > node_ptr->value)
159             {
160                 node_ptr->value = childnode.value;
161                 node_ptr->movepos = affected_list[0];
162             }
163         }
164         /*如果是对手一方，部分回溯值是子结点中最小的一个*/
165         else
166         {
167             if(childnode.value < node_ptr->value)
168             {
169                 node_ptr->value = childnode.value;
170                 node_ptr->movepos = affected_list[0];
171             }
172         }
```

```
173             /* α-β 裁减的判断*/
174             if(parent_ptr)
175             {
176                 if(obcolor != computer_side)
177                 {/*α 裁减*/
178                     if(node_ptr->value <= parent_ptr->value)
179                         return;
180                 }
181             else
182                 {/*β 裁减*/
183                     if(node_ptr->value >= parent_ptr->value)
184                         return ;
185                 }
186             }
187             /*找到下一个可落子的点*/
188             start_pos = affected_list[0]+1;
189             memcpy(&childnode.board, &node_ptr->board,
                sizeof(board_type));
190             num = find_move(&childnode.board, start_pos, obcolor,
                affected_list);
191         }
192         return;
193 }
```

代码解析：上面这段代码就是要点 3 的实现，其中第 147～191 行的整个循环就是 α-β 树的裁剪过程。

14.6　黑白棋游戏的整合测试

本节中，笔者将根据前面的测试用例文档进行黑白棋游戏的整合测试。在这里，笔者只对其中几个主要的测试用例进行演示。

14.6.1　主菜单和界面显示功能测试的演示

这个测试用例主要是测试游戏的菜单和界面显示是否成功，根据测试用例文档，其测试步骤如下所述。

（1）运行黑白棋程序，选中其中的.exe 图标，如图 14.6 所示。

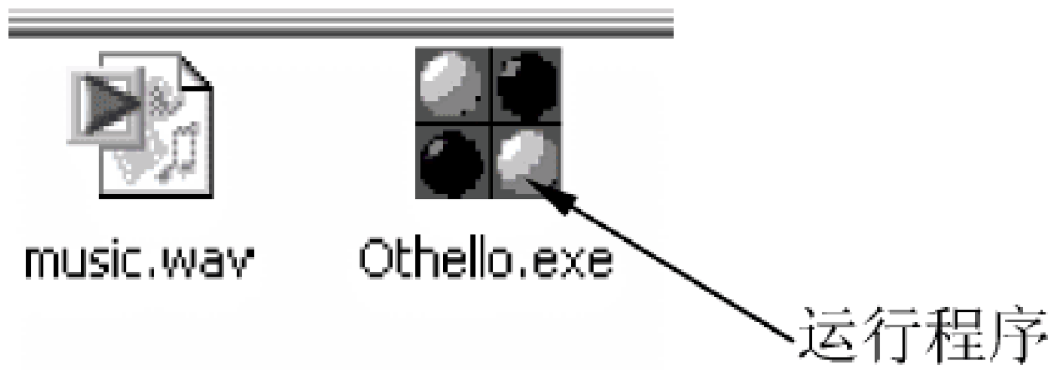


图 14.6　运行黑白棋程序

（2）程序启动后，其菜单及主界面如图 14.7 所示。

判断结果：游戏的菜单和界面显示成功。填写测试用例编号 1.7.1 结果为“通过”。

14.6.2　悔棋功能测试的演示

悔棋功能测试，根据测试用例文档，其测试步骤如下所述。

（1）游戏开始后，与电脑分别落下棋子，如图 14.8 所示。

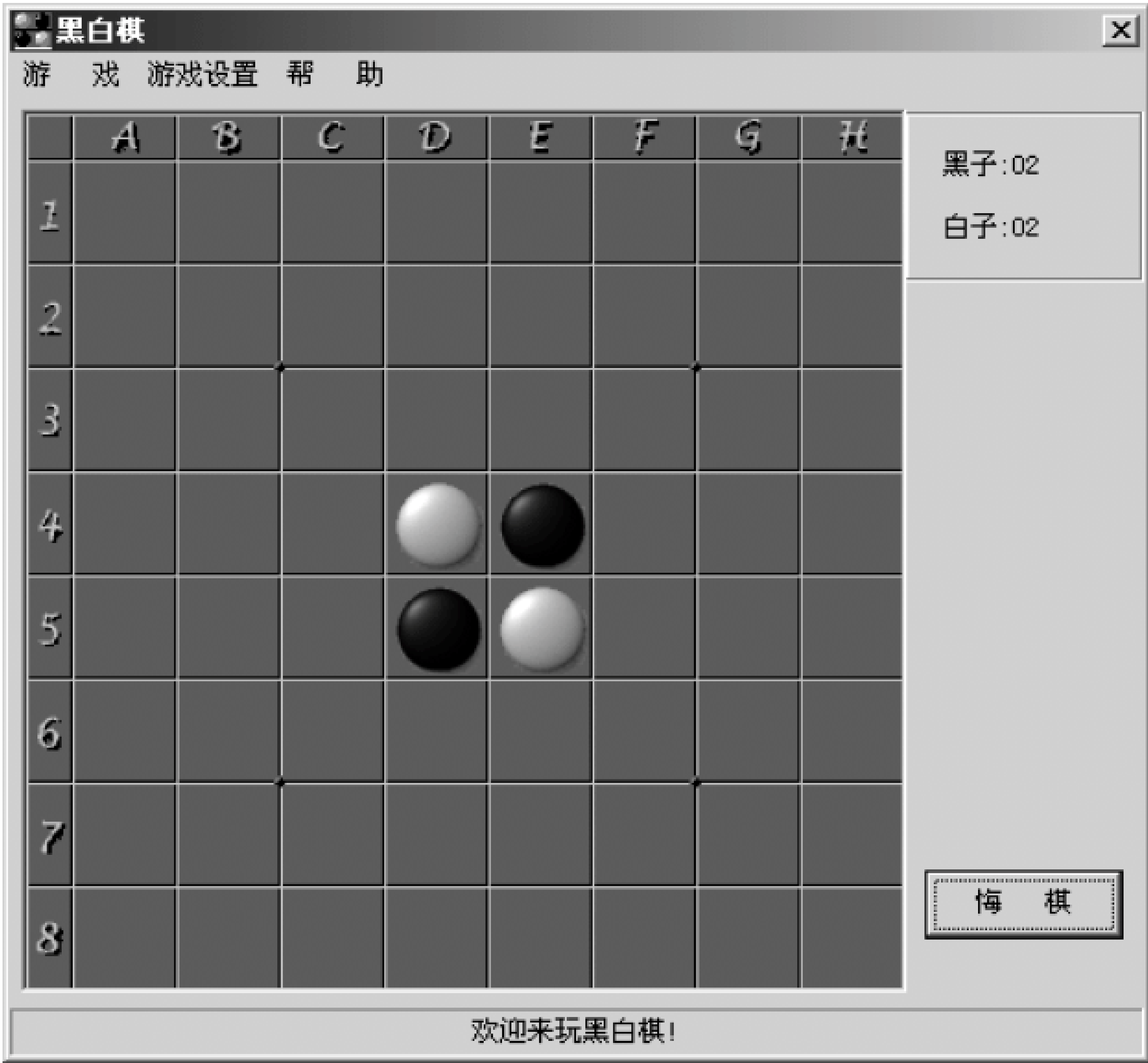


图 14.7　黑白棋游戏主界面及菜单

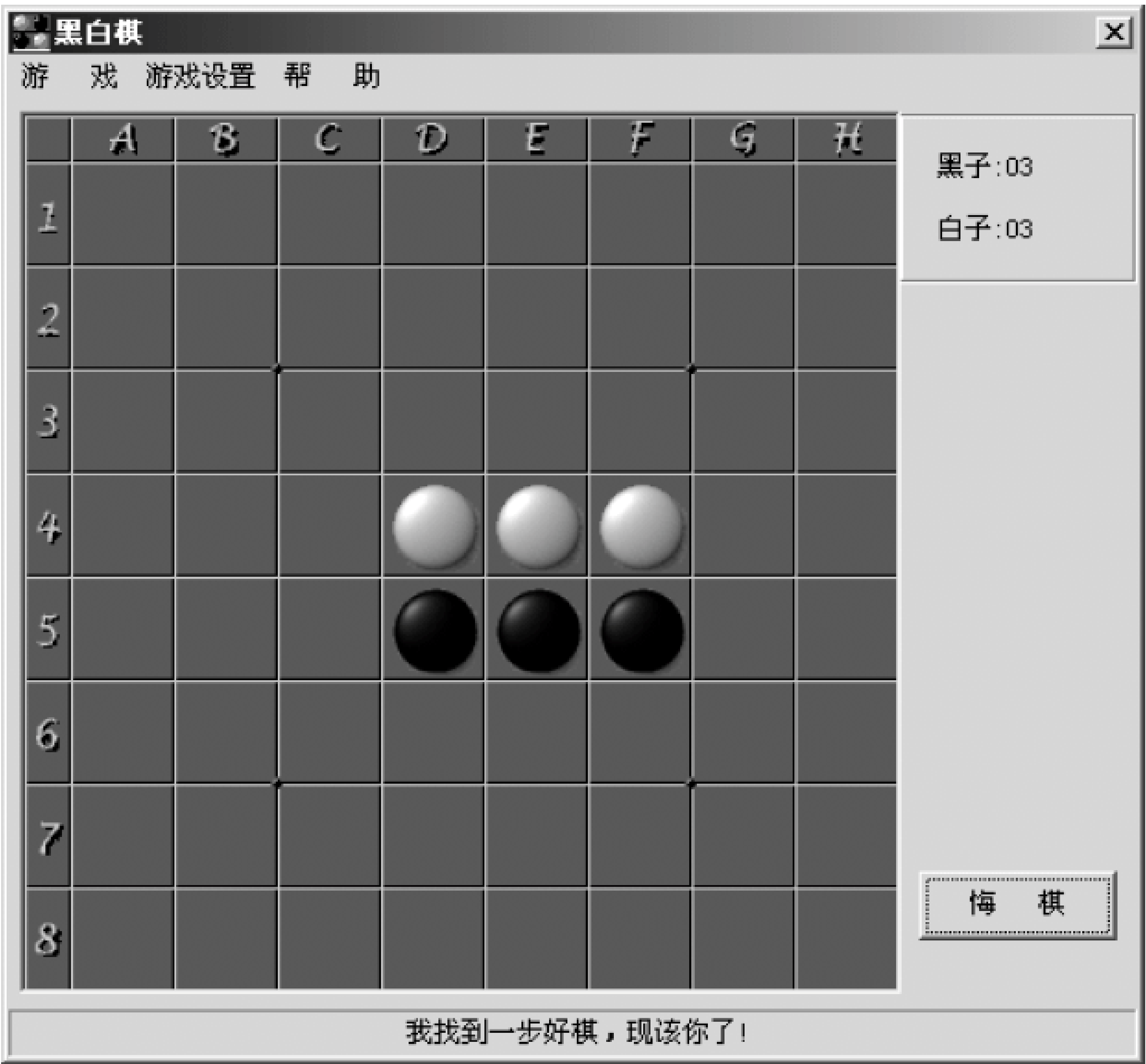


图 14.8　分别落子的棋盘

（2）单击“悔棋”按钮，棋子又恢复成上一步所走的棋盘状态，如图 14.9 所示。

判断结果：悔棋功能测试成功。填写测试用例编号 1.7.2 结果为“通过”。

14.6.3　棋子动画翻转功能测试的演示

测试游戏中的棋子动画翻转功能。根据测试用例文档，其测试步骤如下所述。

（1）翻转前的棋盘格局，如图 14.10 所示。

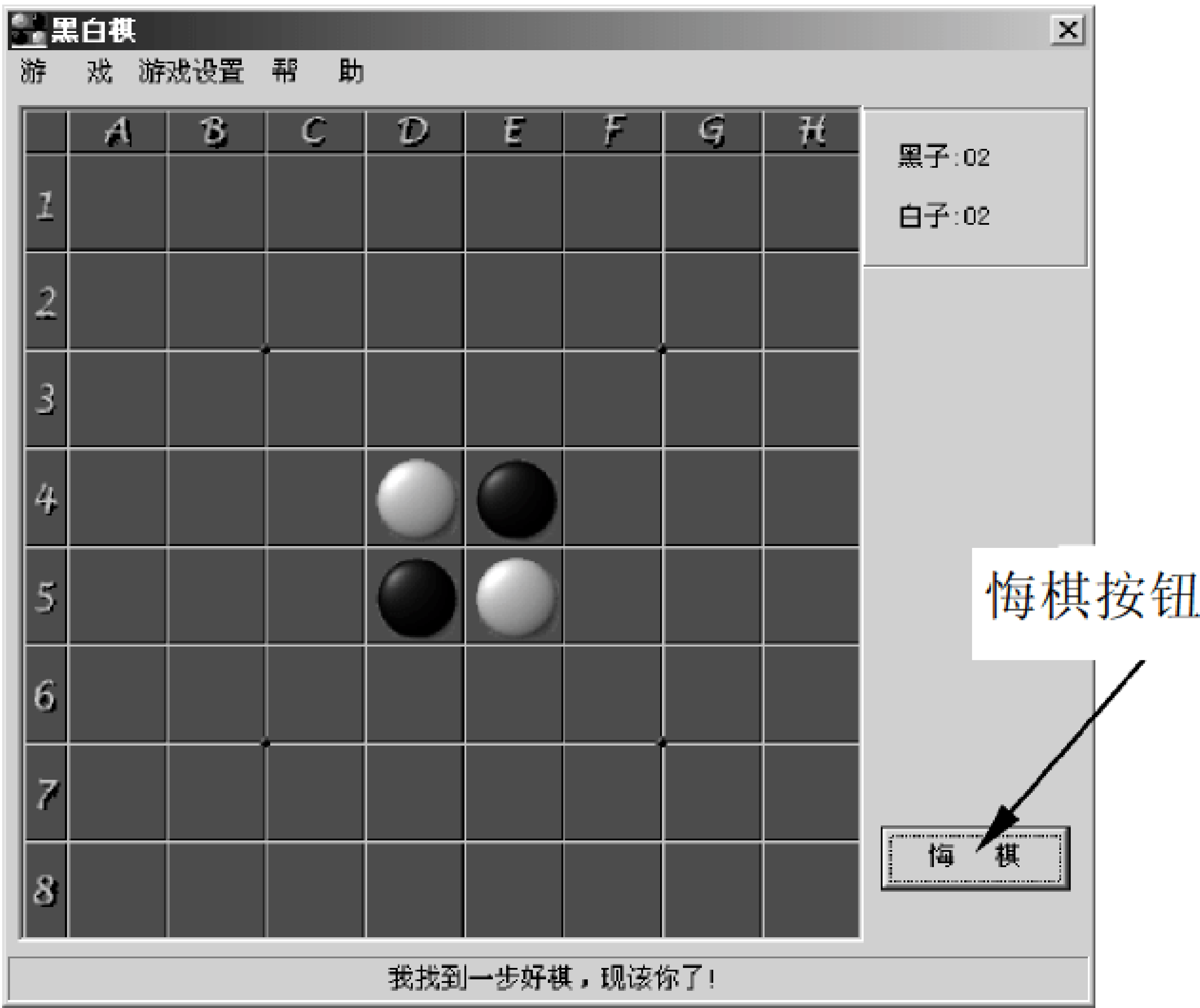


图 14.9　悔棋后的棋盘

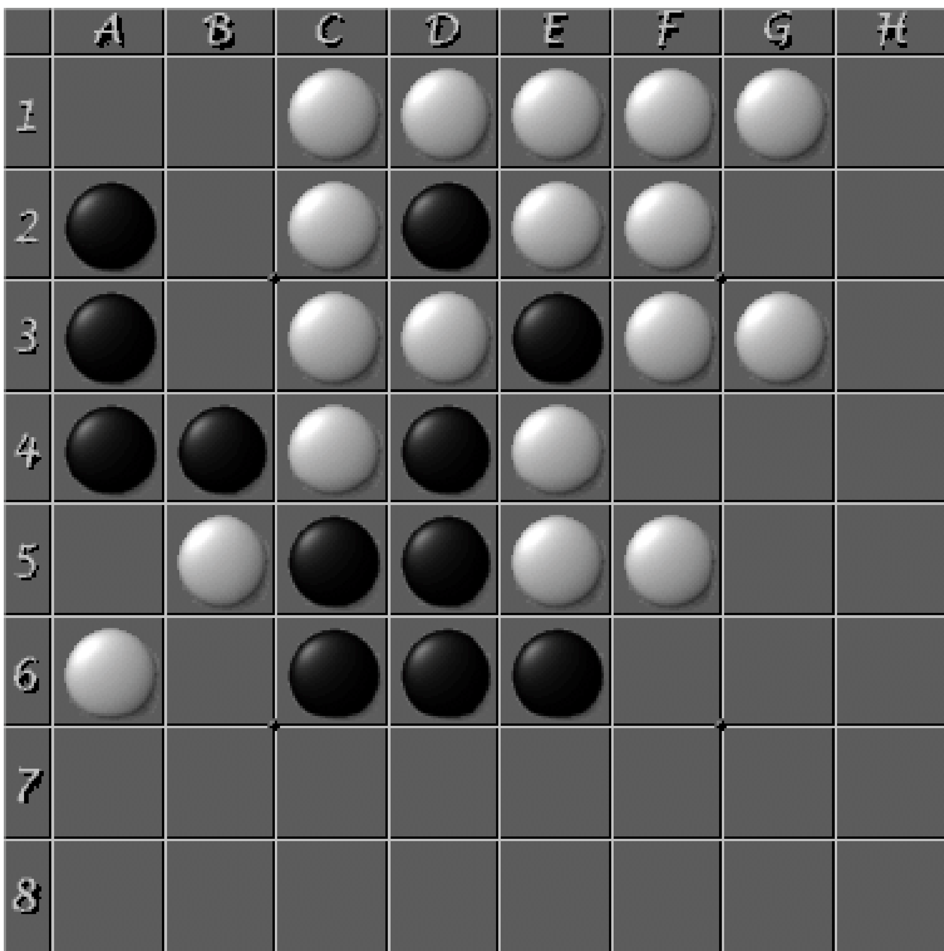


图 14.10　动画翻转前

（2）游戏中的动画翻转过程，如图 14.11 所示。

（3）翻转完成后的棋盘棋局，如图 14.12 所示。

判断结果：通过比较，棋子动画翻转功能正确。填写测试用例编号 1.7.3 结果为“通过”。

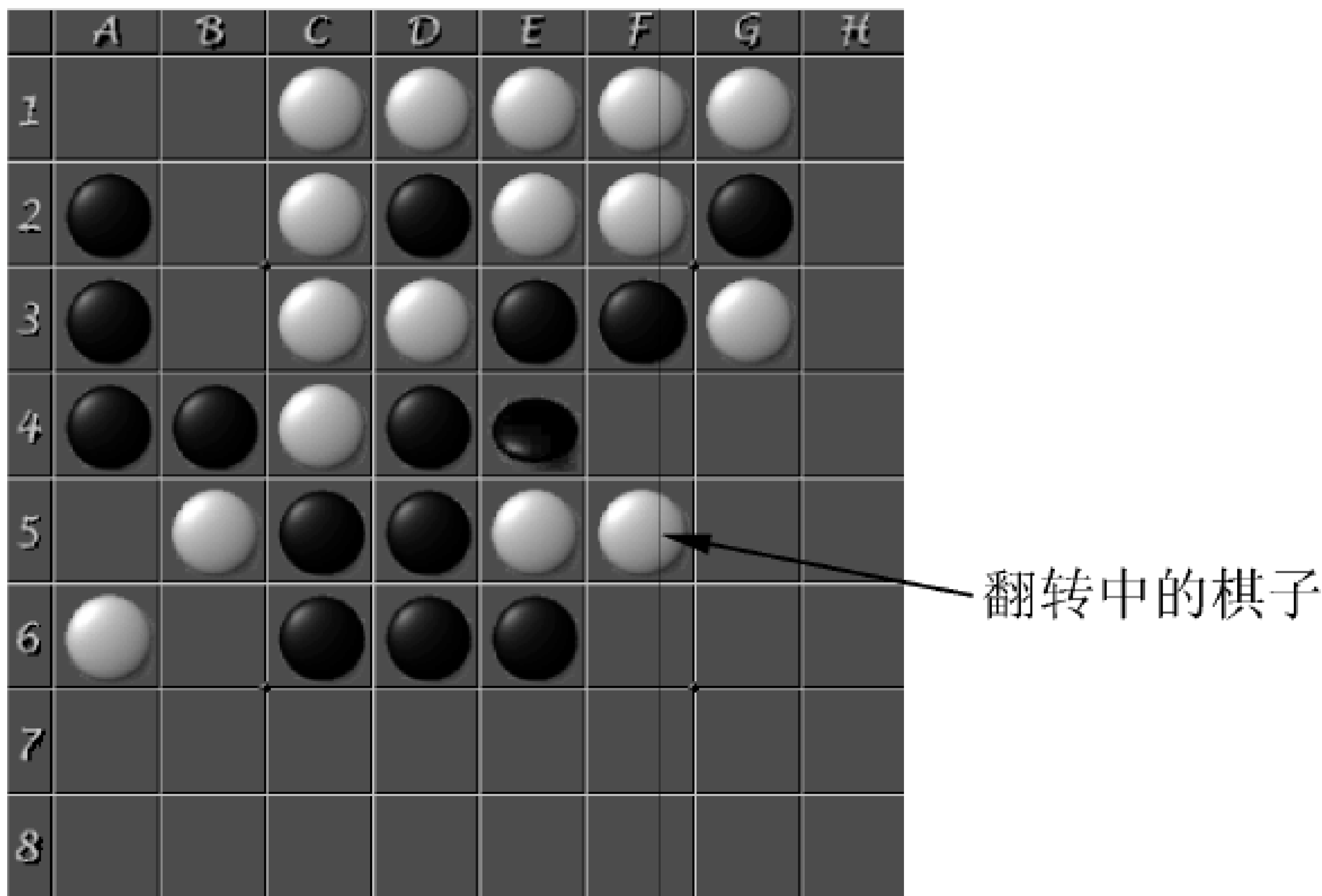


图 14.11　动画翻转中

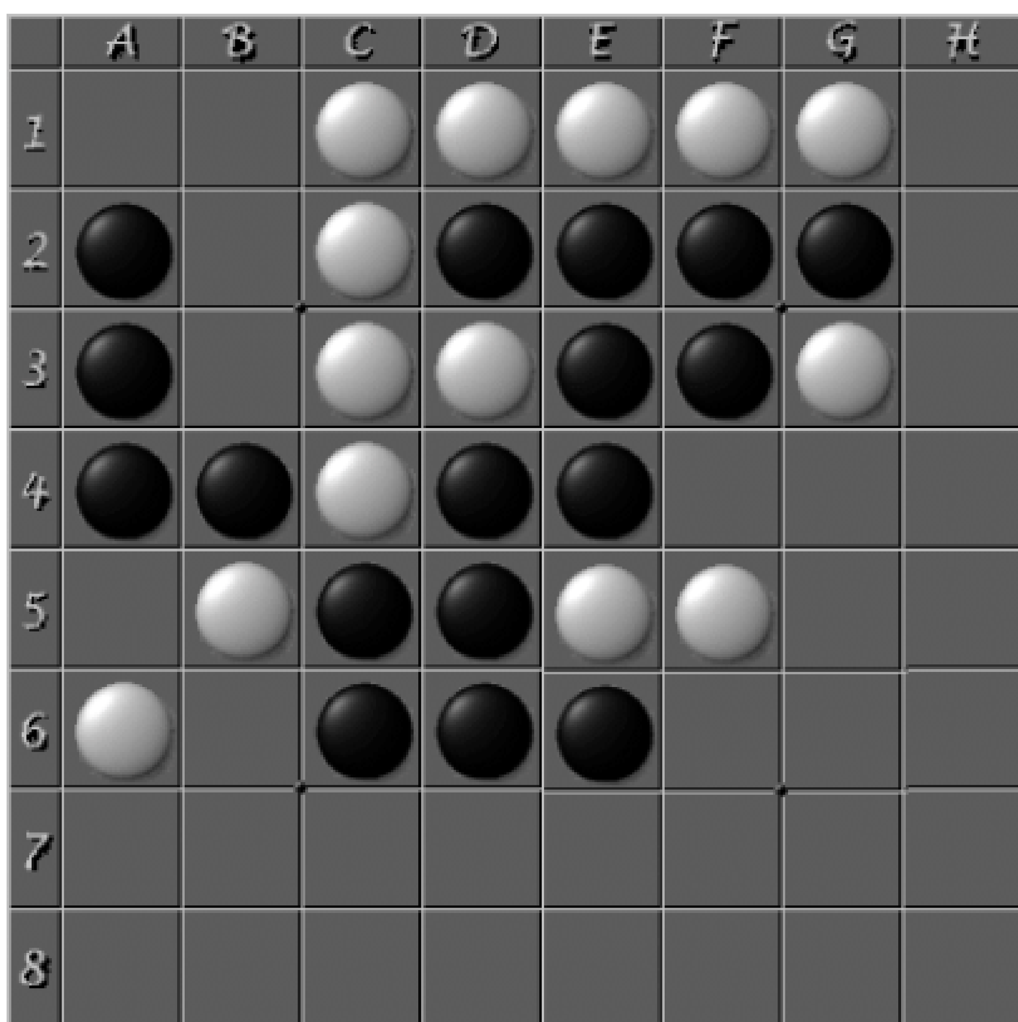


图 14.12　翻转完成后

14.6.4　游戏胜负判断功能测试的演示

测试黑白棋游戏中游戏胜负判断功能。根据测试用例文档，其测试步骤如下所述。

（1）一直交互落子，直到最后一步出来，如图 14.13 所示。

（2）走最后一步，弹出胜负提示，如图 14.14 所示。

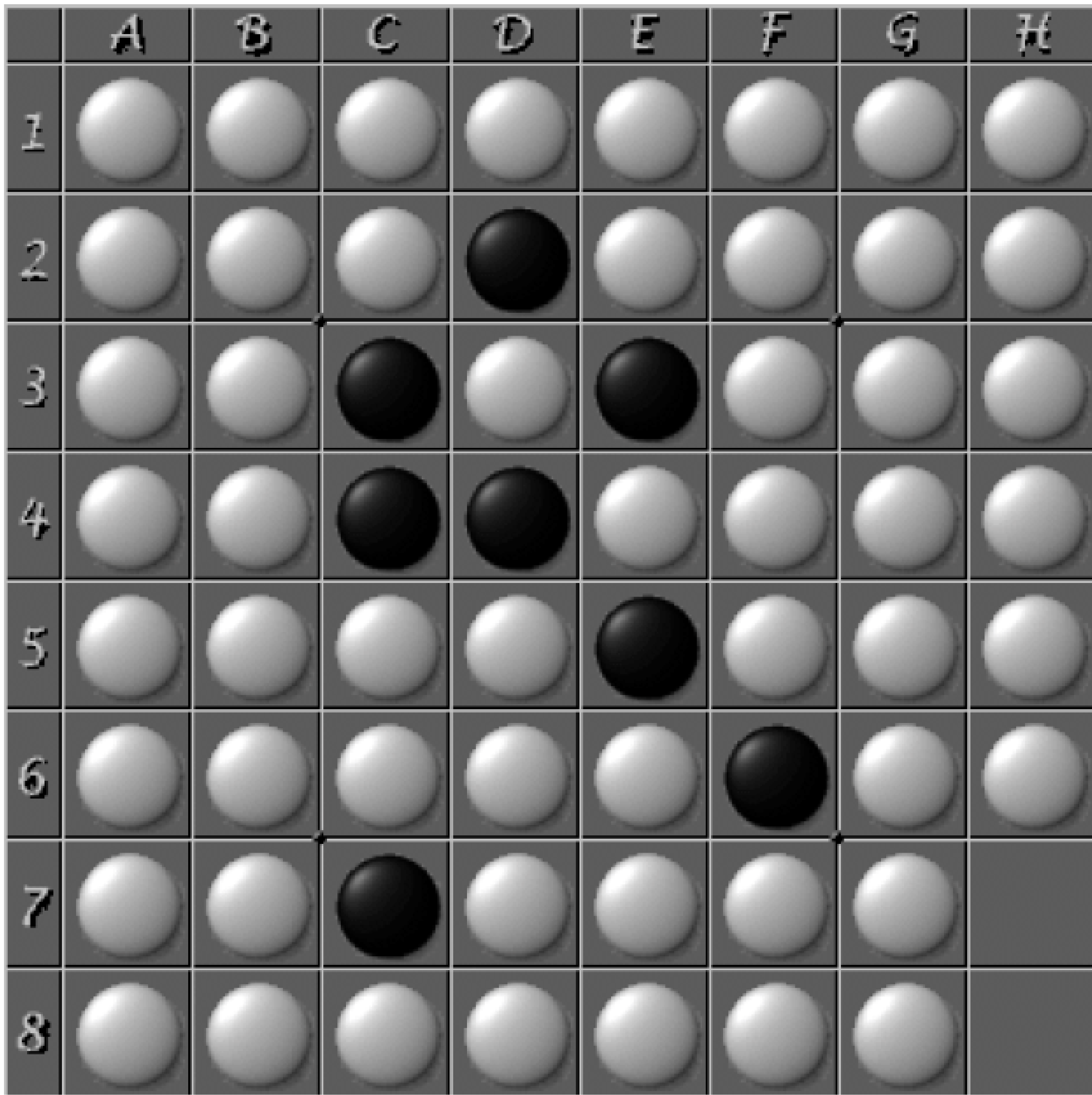


图 14.13　未走最后一步前

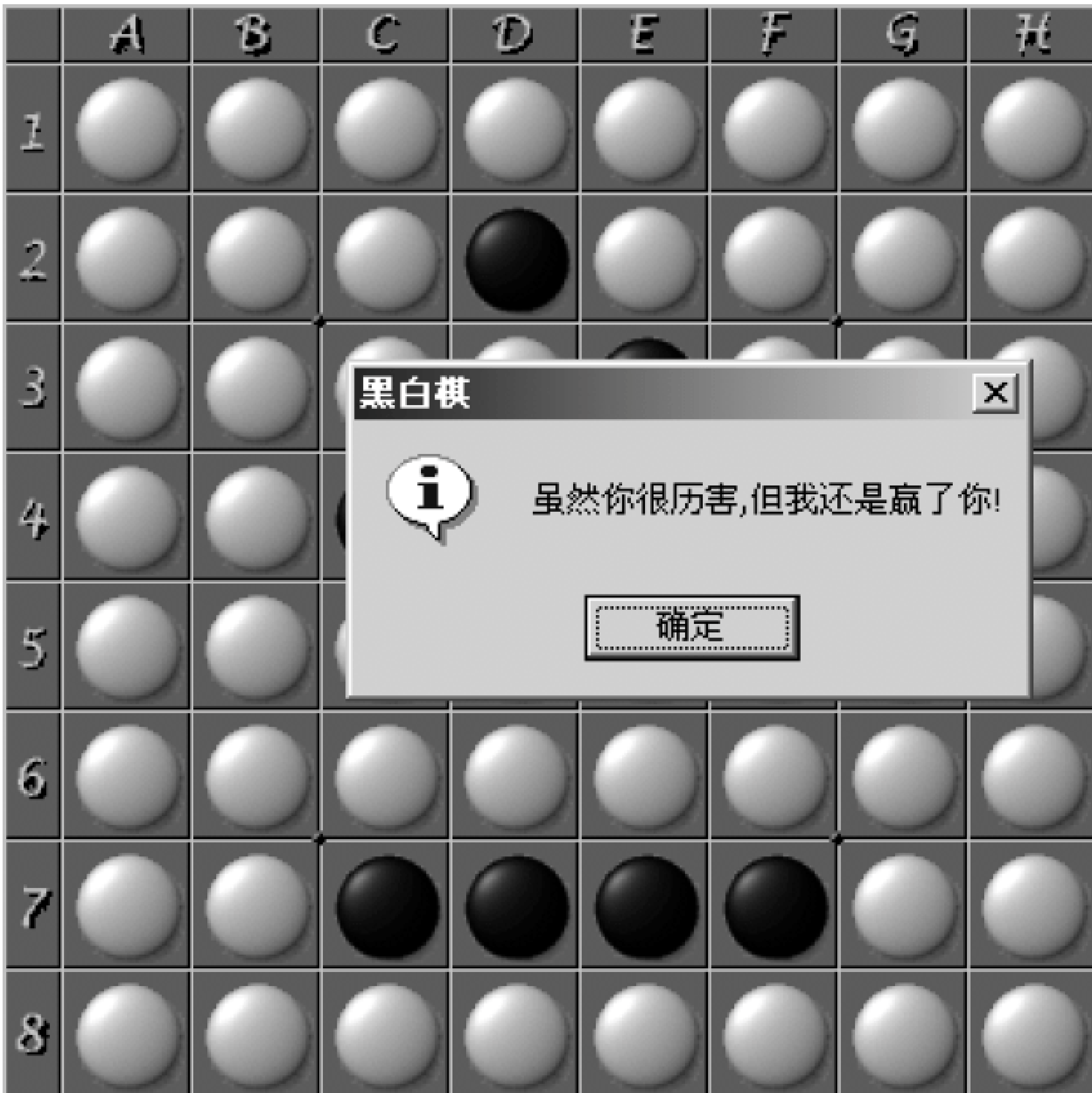


图 14.14　走最后一步后

判断结果：黑白棋游戏中胜负判断功能正确。填写测试用例编号 1.7.4 结果为“通过”。

14.6.5　游戏帮助功能测试的演示

测试黑白棋游戏是否有帮助提示功能。根据测试用例文档，其测试步骤如下：

（1）选择“帮助”|“帮助”命令，如图 14.15 所示。

（2）游戏中弹出“游戏帮助”对话框，如图 14.16 所示。

判断结果：黑白棋游戏帮助提示是正确的。填写测试用例编号 1.7.6 结果为“通过”。

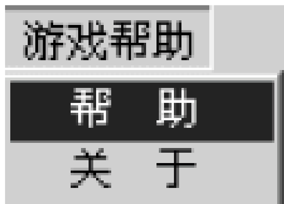


图 14.15　选择“游戏帮助”|“帮助”命令

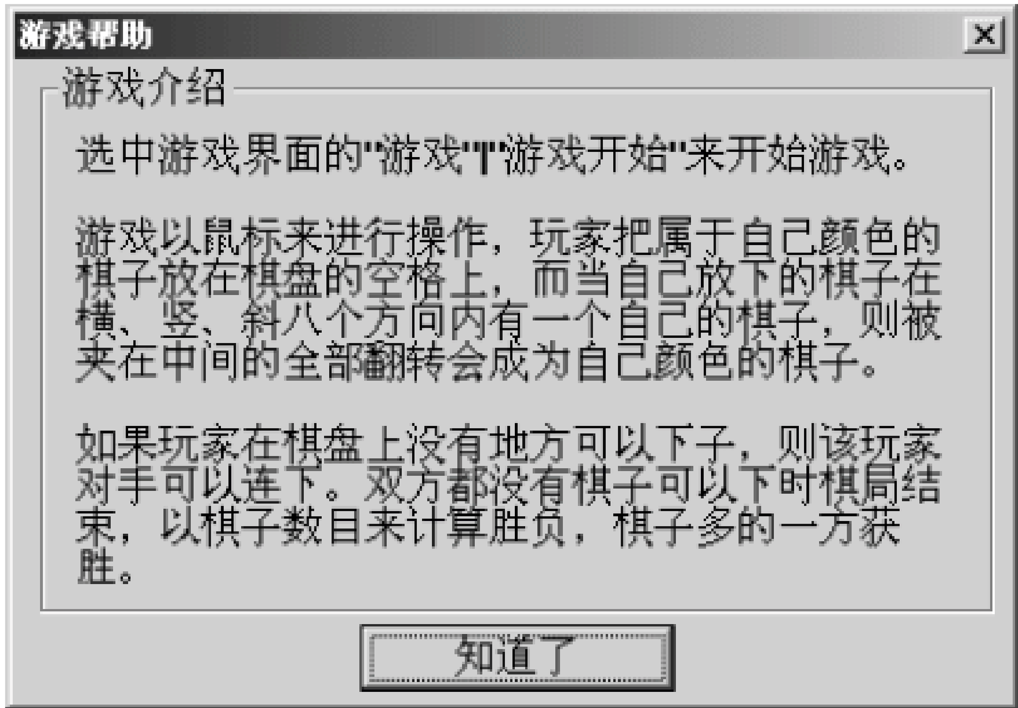


图 14.16　弹出“游戏帮助”对话框

14.7　总　　结

通过本章黑白棋游戏实例项目开发的学习，希望读者能够更加深入地掌握好游戏项目开发中各种文档的格式及编写方法。同时，知道自己如何开发一款黑白棋游戏。本游戏最核心的算法包括：博弈树的构造方法、构造过程的 α-β 裁减、动画翻转算法实现、悔棋算法实现等。这些都是本游戏最难理解的部分，请读者朋友采用一边阅读一边实践的方式来学习，这样才能深刻理解整个算法的核心思想。

第 15 章　扫雷游戏项目开发

学习完第 14 章的黑白棋游戏的开发，本章将继续介绍扫雷游戏实例项目的开发。本章的内容主要是为了帮助读者继续增加项目实际开发的经验，提高对各种文档的编写能力。

本章主要涉及的内容如下：

- 扫雷项目的需求分析。
- 扫雷游戏的概要设计。
- 扫雷游戏操作界面设计。
- 扫雷游戏的详细设计及代码。
- 扫雷游戏的测试用例文档的编写及测试演示。

15.1　扫雷游戏项目的需求分析

获得用户需求并对其进行详细分析，是项目开始的基础。只有获得明确的需求，并做出好的需求分析文档得到客户的认可，才能保证项目的成功。

技巧： 让用户体验初步的软件模型后，再对需求进行修订，比需求一次成型更能贴近用户的真正需求。

15.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到扫雷游戏开发的资料如下所述。

1．扫雷游戏概述

扫雷游戏，是 Windows 操作系统自带的一款经典游戏。其规则简单，上手容易，不论男女老少皆可娱乐。扫雷的目的就是要把所有非地雷的格子揭开即胜利。踩到地雷格子就算失败。

2．扫雷的操作方法

游戏主区域由很多个方格组成。使用鼠标随机点击一个方格，方格即被打开并显示出方格中的数字；方格中数字则表示其周围的 8 个方格隐藏了几颗雷；如果点开的格子为空白格，即其周围有 0 颗雷，则其周围格子自动打开。

例如，方格中出现 1，说明上下左右及斜角合计有一颗雷，依次类推，2 则有 2 颗，3 则有 3 颗。在确实是地雷的方格上点了旗子，就安全了，不是地雷的被点了旗子，后面会

被炸死。在不确定是否是地雷的方格上用右键标示“?”符号，表示怀疑这个格子是地雷。这样格子在自动打开时，被标示的格子就不会被打开，导致游戏结束。

3．扫雷游戏的基本规则

在游戏中，当玩家不小心踩到地雷格子就算失败。而当地雷计数器中的数字变成 0 时，说明地雷全部被查到，游戏结束，玩家胜利。

4．英雄榜的显示及更新

当有玩家把当前等级的地雷全部扫出来后，并且时间不比记录中的时间短。在结束游戏时，要求玩家把名字保存下来。游戏初始时间为 999 秒。

例如，当第一个玩家把全部地雷扫出来的时间为 120 秒，这时玩家的记录时间将被保存下来并作为记录时间线。直到有玩家的时间少于 120 秒，才能更新当前记录时间线并保存玩家的名字。

5．有背景音乐支持

在游戏中，能够选择播放背景音乐。

6．游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

15.1.2　对语言描述进行需求分析

根据《扫雷用户需求描述文档》中的内容，现在需要对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

技巧：在文档中，功能部分一定要明确地提出来，并与用户交流确定是否正确。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类扫雷游戏。特制定本说明书来用于描述某公司扫雷项目开发的功能性需求。

1.1　编写目的

使用技术性语言对某公司的扫雷游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．文档范围

包含某公司扫雷游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司扫雷游戏开发相关的需求分析、程序设计、代码编

写、测试和维护等部门（单位）的人员。

4．参考文献

《扫雷用户需求描述文档》。

5．游戏具有的功能

5.1　能够显示主菜单和界面

游戏需要提供主菜单让玩家进行游戏设置，同时能够显示当前剩余的地雷数量及当前花费时间等相关信息到界面上。

5.2　能够接收鼠标输入功能

能够接收玩家的鼠标输入功能，左键和右键输入。

5.3　能够根据规则翻转相应的格子

翻动指定位置的格子。

5.4　能够标示指定格子的功能

即能够对确定或者怀疑是地雷的格子进行“旗子”和“？”符号的标示。

5.5　游戏胜负判断功能

当玩家单击的格子中有地雷时，判定玩家失败，游戏结束。当扫雷游戏中全部格子被打开时，判定玩家成功。

5.6　英雄榜记录更新

当有玩家把当前等级的地雷全部扫出来后，并且时间比记录中的时间短时，在结束游戏时，要求玩家把名字保存下来。游戏初始时时间为 999 秒。

例如，当第一个玩家把全部地雷扫出来的时间为 120 秒，这时玩家的记录时间将被保存下来并作为记录时间线。直到有玩家的时间少于 120 秒，才能更新当前记录时间线并保存玩家的名字。

5.7　游戏支持背景音乐功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.8　游戏提供帮助说明

在游戏菜单中，提供一个使用说明项，以方便对本游戏不了解的玩家对游戏进行操作和使用。

15.2　扫雷游戏概要设计

笔者编写的扫雷游戏的概要设计文档内容如下所述。

1．引言

1.1　编写目的

为了让每个开发人员明白扫雷游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．术语

3．参考文献

《扫雷游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，确定游戏的最终目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高鼠标及键盘操作的易用性。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。
- Microsoft® Visual C++ 6.0。

4.3　需求概述

内容参见《扫雷游戏需求分析说明书》。

4.4　条件与限制

无。

5．总体设计

5.1　扫雷游戏的功能架构（如图 15.1 所示）

5.2　各功能处理流程

内容参见《扫雷游戏各功能详细设计文档》。

6．接口设计

内容参见《扫雷游戏操作界面设计文档》。

7．程序结构设计

游戏共由 4 个类和一个模块组成，如图 15.2 所示。

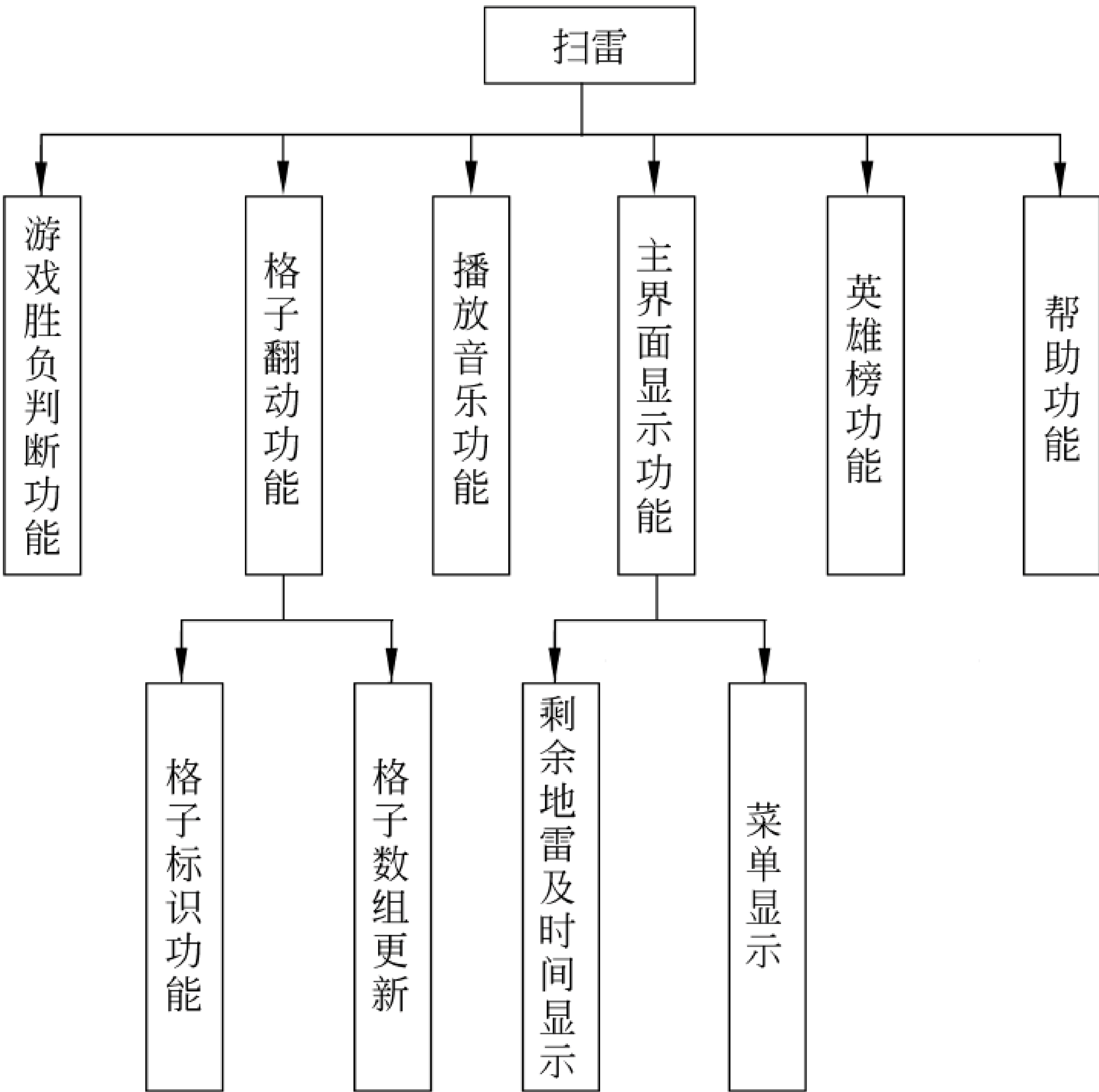


图 15.1　扫雷功能架构

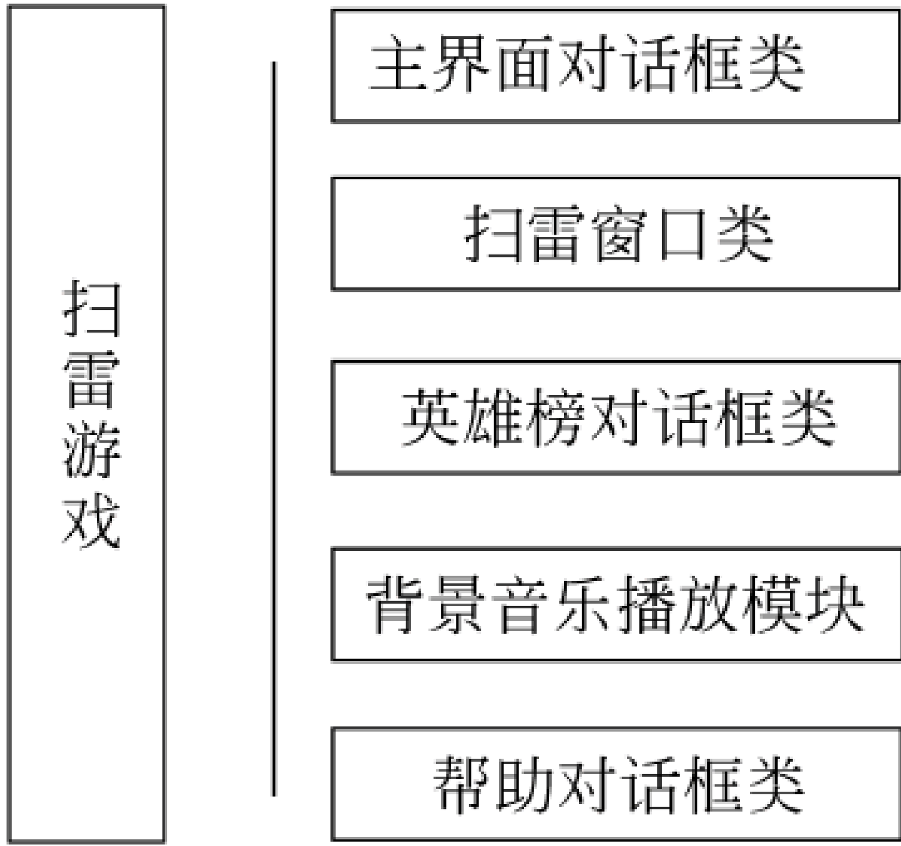


图 15.2　游戏主要结构类

- 主界面对话框类：主要负责主界面、菜单及各个窗口类对象的创建和调用等处理。
- 扫雷窗口类：主要负责接收玩家鼠标输入的打开格子位置、格子变换、花费时间及地雷格子的显示等处理。
- 英雄榜对话框类：主要负责游戏等级记录的更新。
- 背景音乐播放模块：主要负责游戏中背景音乐的播放。
- 帮助对话框类：主要负责帮助提示的显示及其他辅助信息。

8．出错处理设计

8.1　出错输出信息

当游戏中出现错误，采用弹出对话框的方式来提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误，采用中止当前游戏并重新开始新游戏的方法来处理游戏中的错误。

9．维护设计

由于整个扫雷游戏项目在开发完成后，基本不会有太多的变动，所以维护的主要任务是把用户使用中出现的问题解决。

15.3　扫雷游戏操作界面及测试用例设计

本节将继续介绍《游戏操作界面设计文档》和《游戏测试用例文档》的编写。

技巧： 测试用例要以客户语言描述为基础、需求分析文档为辅助来编写，这样才能达到客户和测试人员都满意的目的。

15.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的扫雷游戏的操作界面设计文档如下所述。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确扫雷游戏的操作界面是如何设计的，特制定本文档用于描述本公司扫雷游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．文档范围

包含本公司扫雷游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《扫雷游戏需求分析说明书》；

《扫雷游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面的设计

扫雷的游戏主界面设计如图 15.3 所示。

5.2　游戏菜单结构的设计

扫雷的游戏菜单设计如图 15.4 所示。

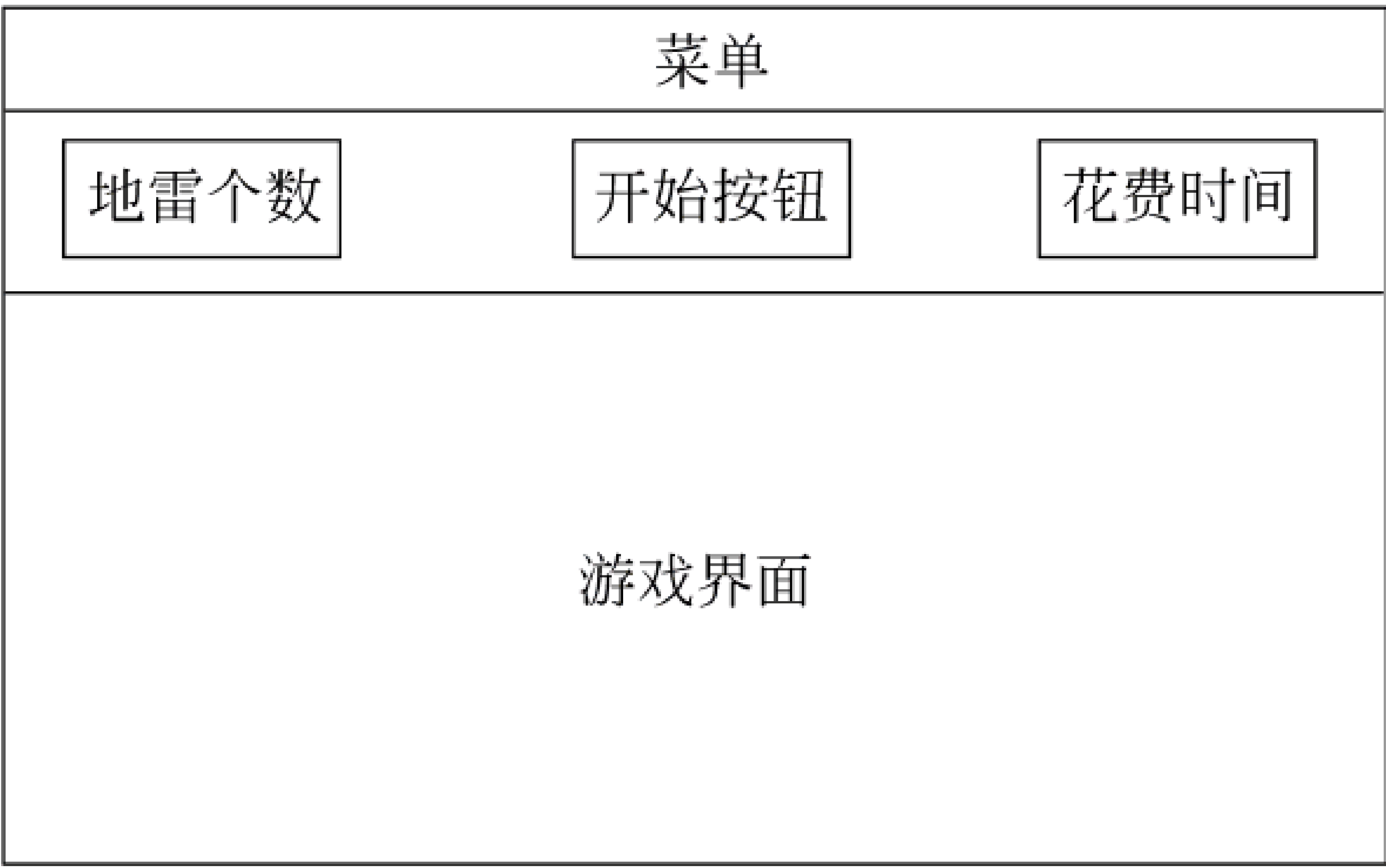


图 15.3　设计的游戏主界面

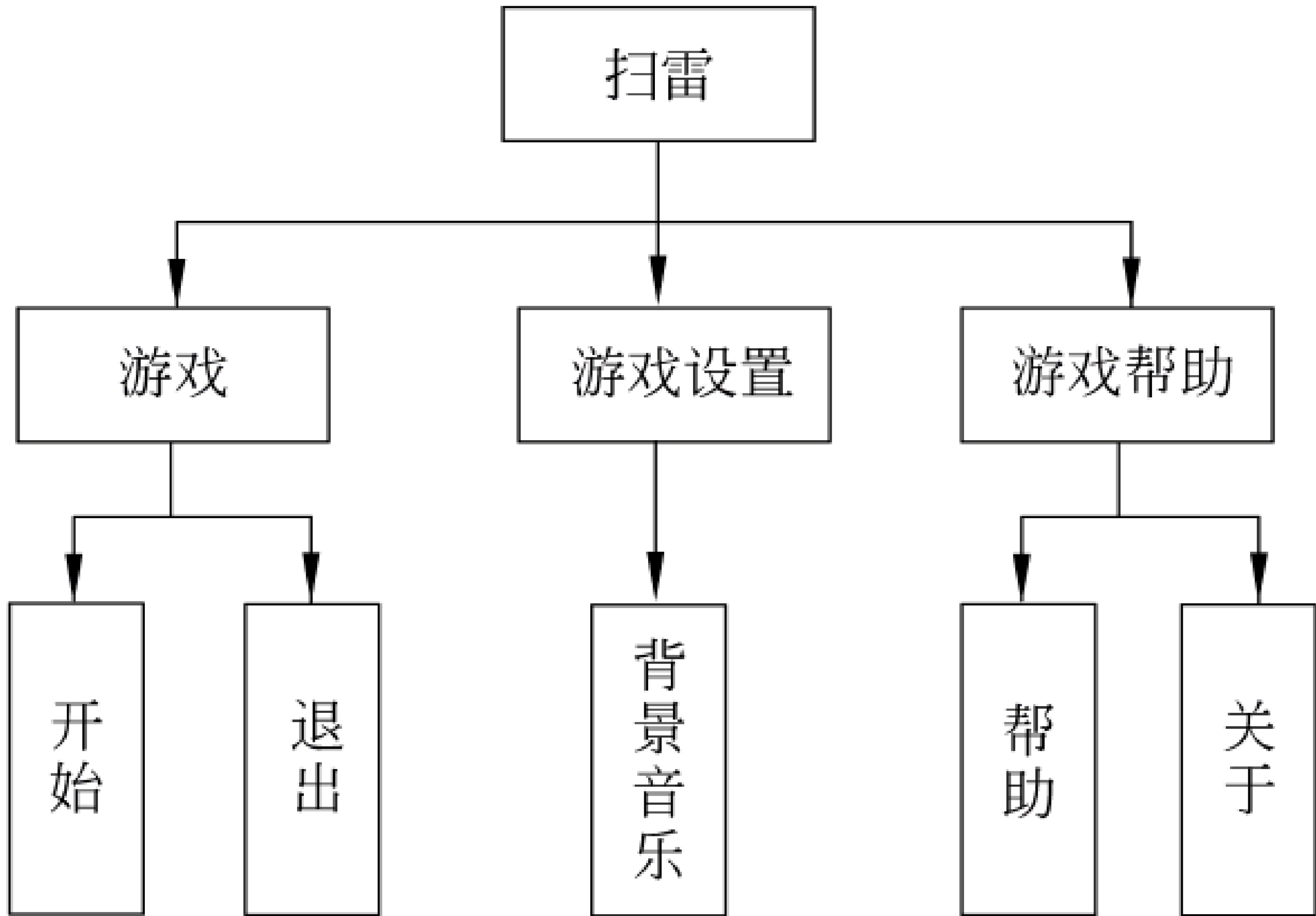


图 15.4　设计的游戏菜单结构

15.3.2　测试用例文档

测试用例文档主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的扫雷游戏测试用例文档内容如下所述。

1．引言

本文档主要用于某公司在开展扫雷游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了扫雷游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过，方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的扫雷游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行扫雷游戏项目测试和验收时使用。

3．使用对象

本测试用例文档使用对象主要是与某公司扫雷游戏开发相关的需求分析、测试和维护

等部门（单位）的人员。

4．参考文献

《扫雷游戏的需求分析说明书》；

《扫雷游戏的概要设计文档》；

《扫雷游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试项目

测试项目主要针对扫雷中各种功能进行整合性测试，共包含如下几个项目。

（1）主菜单和界面显示功能的测试，主要内容如表 15.1 所示。

表 15.1　主菜单和界面显示功能的测试

测试编号：1.7.1	类别：A
项　　目：扫雷测试	
分 项 目：主菜单和界面显示功能的测试	
测试目的：测试扫雷游戏中的菜单和界面是否正确显示	
测试配置：	
预置条件： 扫雷游戏源程序已经编译完成，并可以运行； 键盘和鼠标已准备好	
测试步骤： 运行扫雷程序，查看菜单和界面	
预期结果： 游戏主界面及菜单与操作设计文档中的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏主界面和菜单是否正确显示（是/否）	
测试结果： 通过/不通过	

（2）鼠标输入功能的测试，主要内容如表 15.2 所示。

表 15.2　鼠标输入功能的测试

测试编号：1.7.2	类别：A
项　　目：扫雷测试	
分 项 目：鼠标输入功能的测试	
测试目的：测试游戏是否支持鼠标的左键和右键输入功能	
测试配置：	
预置条件： 扫雷游戏已经开始； 鼠标已经准备好	

续表

测试步骤： 使用鼠标左键单击主游戏窗口中的格子； 使用鼠标右键单击主游戏窗口中的格子
预期结果： 当使用左键时，指定的格子被打开； 当使用右键时，指定的格子被标示
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏是否支持鼠标输入功能（是/否）
测试结果： 通过/不通过

（3）标示指定格子功能的测试，其主要内容如表 15.3 所示。

表 15.3　标示指定格子功能的测试

测试编号：1.7.3	类别：A
项　　目：扫雷测试	
分 项 目：标示指定格子功能的测试	
测试目的：测试是否能够对指定游戏中格子进行标示	
测试配置：	
预置条件： 扫雷游戏已经开始； 鼠标已经准备好	
测试步骤： 在窗口中的任意格子上右击	
预期结果： 第一次单击：在指定格子的上面出现相应的“旗子”符号； 第二次单击：在指定格子的上面出现相应的“？”符号	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中是否能够对指定游戏中格子进行标示（是/否）	
测试结果： 通过/不通过	

（4）游戏胜负判断功能的测试，主要内容如表 15.4 所示。

表 15.4　游戏胜负判断功能的测试

测试编号：1.7.4	类别：A
项　　目：扫雷测试	
分 项 目：游戏胜负判断功能的测试	

续表

测试目的：测试游戏能否给出正确的胜负判断
测试配置：
预置条件： 游戏已经开始； 鼠标准备好
测试步骤： 当前游戏中的地雷已经全部被查到，地雷个数为 0； 有一个地雷格子被反转出来
预期结果： 玩家胜利； 玩家失败
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 当胜利或者失败时，游戏能否给出正确的判断（是/否）
测试结果： 通过/不通过

（5）背景音乐播放功能的测试，主要内容如表 15.5 所示。

表 15.5　背景音乐播放功能的测试

测试编号：1.7.5	类别：A
项　　目：扫雷测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试扫雷游戏能否支持播放背景音乐	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否支持播放背景音乐（是/否）	
测试结果： 通过/不通过	

（6）帮助功能的测试，主要内容如表 15.6 所示。

表 15.6　帮助功能的测试

测试编号：1.7.6	类别：A
项　　目：扫雷测试	
分 项 目：帮助功能的测试	
测试目的：测试扫雷游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选择“游戏帮助”|“帮助”命令	
预期结果： 出现游戏帮助提示，说明游戏操作方法	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 扫雷游戏是否有帮助提示功能（是/否）	
测试结果： 通过/不通过	

15.4　扫雷游戏的界面实现

扫雷游戏的 Visual C++工程采用 MFC 对话框模式进行开发。本节主要讲解扫雷游戏各个功能模块的代码实现。

说明： 界面是提高用户友好度最直接的方法，所以界面设计是游戏开发中重要的部分。

15.4.1　游戏菜单的实现

在扫雷游戏中，通过如下几步即可实现添加游戏的菜单。

（1）在扫雷游戏工程的资源中添加一个菜单资源，其属性如表 15.7 所示。

表 15.7　主菜单属性

ID	类　　别	说　　明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
IDR_EXIT_GAME	菜单栏	退出游戏
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于

（2）给每个菜单栏添加响应函数到 CMineDlg 类中。

（3）菜单响应函数的实现，如代码 15.1 所示。

代码 15.1　菜单响应函数的实现

```
01  BEGIN_MESSAGE_MAP(CMineDlg, CDialog)
02      ON_WM_SYSCOMMAND()
03      ON_WM_PAINT()
04      ON_WM_QUERYDRAGICON()
        // 菜单资源与函数映射表
05      ON_COMMAND(IDR_ABOUT, OnAbout)
06      ON_COMMAND(IDR_EXIT_GAME, OnExitGame)
07      ON_COMMAND(IDR_HELP, OnHelp)
08      ON_COMMAND(IDR_PLAY_MUSIC, OnPlayMusic)
09      ON_COMMAND(IDR_START_GAME, OnStartGame)
10  END_MESSAGE_MAP()
11
12  BOOL CMineDlg::OnInitDialog()              //初始化对话框
13  {
14      CDialog::OnInitDialog();
15
16      m_bStart = FALSE;                      //设置游戏状态
17
18      InitMenu();                            //初始化菜单
19
20      return TRUE;                           //初始化成功
21  }
22  ...                                        //省略部分代码
23  void CMineDlg::OnOK()                      //单击“确认”按钮响应函数
24  {
25      CDialog::OnOK();
26  }
27
28  void CMineDlg::OnCancel()                  //单击“退出”按钮响应函数
29  {
30      CDialog::OnCancel();
31  }
32
33  void CMineDlg::OnAbout()                   //关于菜单栏响应函数
34  {
35      CAboutDlg dlg;                         //创建关于对话框
36      dlg.DoModal();                         //弹出关于对话框
37  }
38
39  void CMineDlg::OnExitGame()                //退出菜单栏响应函数
40  {
41      CDialog::OnCancel();                   //调用基类退出函数
42  }
43
44  void CMineDlg::OnHelp()                    //帮助菜单栏响应函数
45  {
46      CHelpDlg dlg;                          //创建帮助对话框
47      dlg.DoModal();                         //弹出帮助对话框
48  }
49
50  void CMineDlg::OnPlayMusic()               //背景音乐菜单栏响应函数
51  {
52      CWnd*  pMain  =  AfxGetMainWnd();
```

```
53      CMenu*   pMenu    =   pMain->GetMenu();
54      //判断播放音乐菜单当前状态
55     BOOL bCheck = (BOOL)pMenu->GetMenuState(IDR_PLAY_MUSIC, MF_CHECKED);
56
57      if(m_bStart)
58      {
59          if(bCheck)
60          {
61              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
62                  MF_BYCOMMAND | MF_UNCHECKED);
63          }
64          else
65          {
66              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
67                  MF_BYCOMMAND | MF_CHECKED);
68          }
69
70          PlayBackMusic(!bCheck);          //调用播放背景音乐功能函数
71      }
72  }
73
74  void CMineDlg::OnStartGame()
75  {
76      GameStart();                         //调用游戏开始接口函数
77  }
78
79  void CMineDlg::InitMenu()                //初始化菜单函数
80  {
81      CWnd*   pMain    =   AfxGetMainWnd();
82      CMenu*   pMenu    =   pMain->GetMenu();
83      pMenu->CheckMenuItem(IDR_PLAY_MUSIC,MF_BYCOMMAND| MF_UNCHECKED);
84  }
```

代码解析：代码第 79 行，初始化菜单函数的实现，主要实现如何通过窗体指针操作指定菜单的状态。

15.4.2　游戏帮助对话框的实现

扫雷游戏中的帮助是使用一个对话框来实现的。其实现步骤如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 15.5 所示。

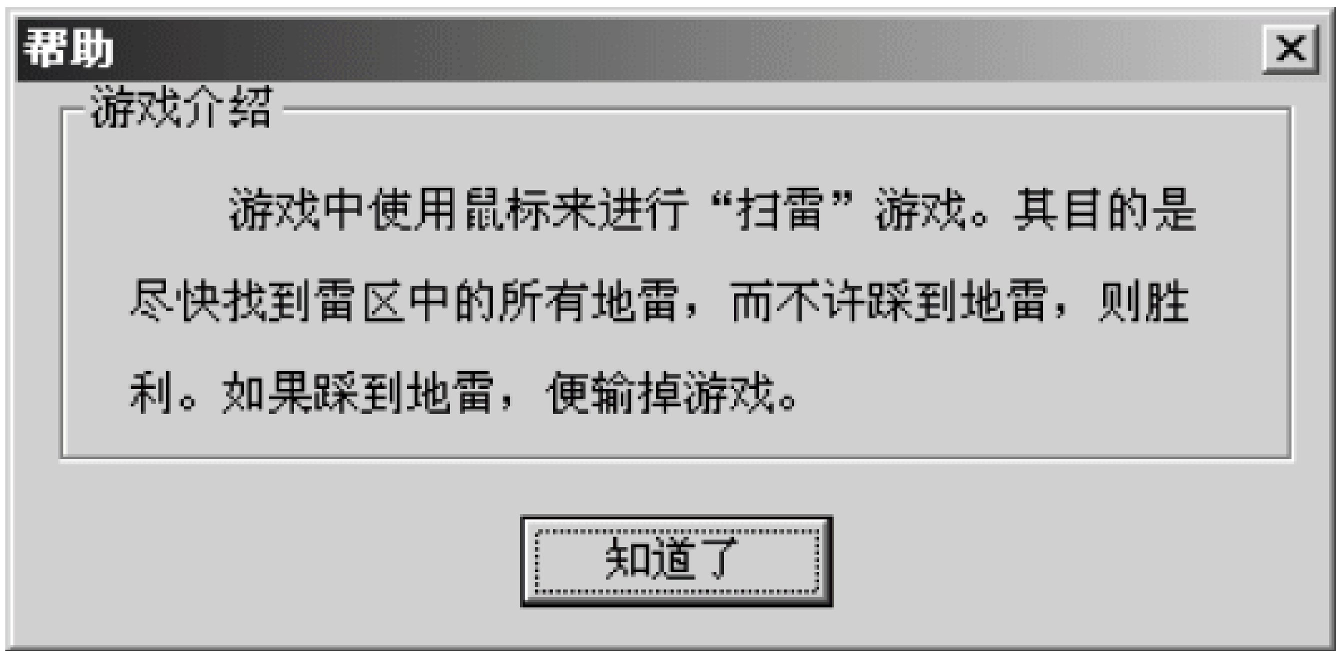


图 15.5　帮助对话框

（2）编写一个 CHelpDlg 对话框类，主要是加载 IDD_HELP 对话框资源。通过资源中的文字说明对游戏操作方法进行描述。同时只包含单击“知道了”按钮的响应函数。其类声明如代码 15.2 所示。

代码 15.2　CHelpDlg 对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
```

```
04  // HelpDlg.h CHelpDlg类声明头文件
05
06
07  ////////////////////////////////////////////////////////////////////
08  // CHelpDlg 对话框类
09
10  class CHelpDlg : public CDialog        //公共继承于CDialog类
11  {
12  public:
13      CHelpDlg(CWnd* pParent = NULL);     //构造函数
14
15  //对话框资源
16      enum { IDD = IDD_HELP };            //加载资源
17
18  //重载函数
19      protected:
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                //单击“确定”按钮响应函数声明
25      DECLARE_MESSAGE_MAP()
26  };
27
28  #endif
```

（3）CHelpDlg 对话框类的实现，需要实现对话框类的构造函数、析构函数和“知道了”按钮响应函数，其代码如代码 15.3 所示。

代码 15.3　CHelpDlg 对话框类的实现

```
01  // HelpDlg.cpp CHelpDlg类的实现源文件
02
03
04  #include "stdafx.h"                     //插入头文件
05  #include "Othello.h"
06  #include "HelpDlg.h"                    //插入类声明头文件
07
08  //////////////////////////////////////////////////////////////////
09  // CHelpDlg 对话框类实现
10
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/)//构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);
19  }
20
21  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)
22  END_MESSAGE_MAP()
23
24  ////////////////////////////////////////////////////////////
25  // CHelpDlg 消息响应函数
26
27  void CHelpDlg::OnOK()                   //单击“知道了”按钮响应函数
```

```
28   {
29       CDialog::OnOK();
30   }
```

代码解析：上段代码实际上都是由 VC 工具自动生成的。主要功能是将帮助界面显示出来方便用户查看。代码第 27 行，是对“知道了”按钮的响应函数的实现。本代码调用了 CDialog 类的 OnOk()函数实现退出功能。

15.4.3　游戏英雄榜对话框的实现

扫雷游戏英雄榜的实现，分为如下几个步骤。

（1）创建一个对话框资源，并添加相应的控件，如图 15.6 所示。

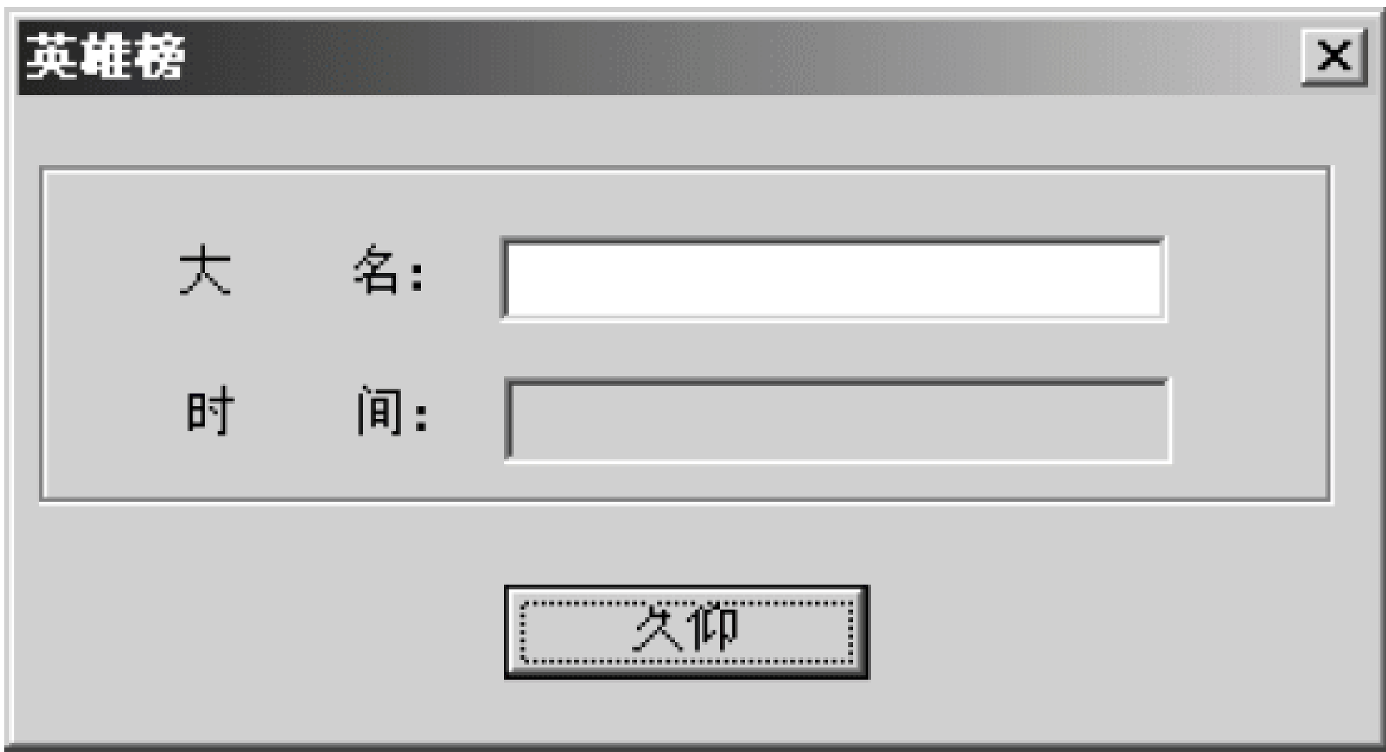


图 15.6　英雄榜对话框资源

（2）配置（setup.ini）文件格式如下：

```
[HERO]
name=XXX
time=0
```

（3）添加 CHeroDlg 类，其中需要包含 IDD_HERO_DLG 对话框资源和“设置可写记录标志”接口函数声明，用于外部函数调用时，设置是否对配置文件进行写操作。类的声明如代码 15.4 所示。

代码 15.4　CHeroDlg 类的声明

```
01   #if !defined(AFX_HERODLG_H__)
02   #define AFX_HERODLG_H__
03
04   // HeroDlg.h 头文件
05
06   ////////////////////////////////////////////
07   // CHeroDlg dialog
08
09   class CHeroDlg : public CDialog
10   {
11   public:
12       void SetWriteFlg(BOOL bflg);              //接口函数，设置可记录标志变量
13       CHeroDlg(CWnd* pParent = NULL);           //构造函数
14
15       enum { IDD = IDD_HERO_LIST };             //对话框资源
16       int      m_time;                          //保存时间变量
17       CString m_name;                           //保存姓名变量
18
```

```
19      public:
20      virtual int DoModal();                    //弹出对话框函数声明
21      protected:
22      virtual void DoDataExchange(CDataExchange* pDX);
23
24  protected:
25
26      virtual void OnOK();                      //单击"久仰"按钮响应函数声明
27
28      DECLARE_MESSAGE_MAP()
29  private:
30      BOOL m_bWriteflg;                         //记录标志变量
31  };
32
33  #endif
```

（4）CHeroDlg 类的实现中通过调用系统 API 函数，来对配置文件进行读写操作。而“设置读写标志”接口函数，是对类的一个成员变量 m_bWrite 进行赋值操作，达到写入或者读取的区分。其代码如代码 15.5 所示。

代码 15.5　CHeroDlg 类的实现

```
01  // HeroDlg.cpp 源文件
02  #include "stdafx.h"                       //插入头文件
03  #include "mine.h"
04  #include "HeroDlg.h"                      //插入类声明头文件
05
06  ///////////////////////////////////////////////
07  // CHeroDlg 对话框
08
09  CHeroDlg::CHeroDlg(CWnd* pParent /*=NULL*/)//构造函数
10      : CDialog(CHeroDlg::IDD, pParent)
11  {
12      m_bWriteflg = FALSE;                  //初始化写标志变量为假
13  }
14
15  void CHeroDlg::DoDataExchange(CDataExchange* pDX)
16  {
17      CDialog::DoDataExchange(pDX);         //变量与资源映射
18      //{{AFX_DATA_MAP(CHeroDlg)
19      DDX_Text(pDX, IDC_TIME_EDIT, m_time);
20      DDX_Text(pDX, IDC_NAME_EDIT, m_name);
21      //}}AFX_DATA_MAP
22  }
23
24  BEGIN_MESSAGE_MAP(CHeroDlg, CDialog)
25      ON_BN_CLICKED(IDOK_BTN, OnBtn)        //按钮与函数映射
26  END_MESSAGE_MAP()
27
28  ////////////////////////////////////////////////////
29  // CHeroDlg 消息句柄
30
31  void CHeroDlg::SetWriteFlg(BOOL bflg)     //设置写入标志
32  {
33      m_bWriteflg = bflg;
34  }
35
36  int CHeroDlg::DoModal()                   //弹出对话框
```

```
37  {
38      char pszTmp[128] = {0};
39
40      //读取配置文件
41      GetPrivateProfileString("HERO", "name", "无名氏",
42          pszTmp, 127, ".\\hero.ini");        //读入姓名
43      m_name = CString(pszTmp);
44
45      if(!m_bWriteflg)
46      {
47          GetPrivateProfileString("HERO", "time", "0",
48              pszTmp, 127, ".\\hero.ini");    //读入等级
49          m_time = atoi(pszTmp);
50      }
51
52      return CDialog::DoModal();
53  }
54
55  void CHeroDlg::OnBtn()                      //按钮响应
56  {
57      UpdateData(TRUE);
58      if(m_bWriteflg)
59      {
60          CString tmp;
61          //写入姓名和时间记录
62          WritePrivateProfileString("HERO", "name", m_name,
            ".\\hero.ini");
63          tmp.Format("%d", m_time);
64          WritePrivateProfileString("HERO", "time", tmp, ".\\hero.ini");
65      }
66      m_bWriteflg = FALSE;
68      CDialog::OnOK();
69  }
70
71  BOOL CHeroDlg::OnInitDialog()               //初始化对话框
72  {
73      CDialog::OnInitDialog();
74
75      if(m_bWriteflg)
76      {                                       //当为写入时，把按钮名称改变
77          SetDlgItemText(IDOK_BTN, "记录");
78      }
79
80      return TRUE;
81  }
```

代码解析：代码第 75 行，是在初始化时根据 m_bWriteflg 的状态设置按钮的名称。代码第 58 行，同样根据这个状态标志，实现配置文件中英雄名称的写入操作。

15.4.4　游戏播放背景音乐的实现

播放游戏背景音乐，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。要实现这个功能，需要如下几个步骤。

（1）在工程文件中，添加 winmm.lib 静态库文件及头文件，参见第 5.4 节。

（2）实现 CMineDlg 类中的 PlayBackMusic()成员函数，其代码如代码 15.6 所示。

代码 15.6　CMineDlg 类的 PlayBackMusic 成员函数实现

```
01  #include <mmsystem.h>                          //插入系统 API 头文件
02  ...
03  void CMineDlg::PlayBackMusic(BOOL bCheck)
04  {
05      //指定文件并播放
06      if(bCheck)
07      {                                          //播放指定音乐文件
08          sndPlaySound("music.wav",SND_ASYNC);
09      }
10      else
11      {                                          //停止播放
12          sndPlaySound(NULL,SND_PURGE);
13      }
14  }
```

15.5　扫雷游戏的核心算法设计与实现

在前面的章节中已经讲解了扫雷游戏的菜单和各种对话框的实现。本节将对扫雷游戏的核心算法的设计和实现进行讲解。

15.5.1　新游戏处理模块的设计与实现

新游戏处理模块主要负责游戏中的游戏初始化及开始游戏。其设计比较简单，只需要通过如下几个步骤即可实现。

（1）载入图片资源和配置文件中的数据。

（2）把所有的游戏参数进行初始化。例如，当前消耗时间和状态等。

（3）初始化表示地雷区域的二维数组。

（4）让地雷区域图像失效，重新绘制新的图像。

其实现如代码 15.7 所示。

代码 15.7　新游戏处理模块的实现

```
01  /*配置文件读取*/
02  void CMyMine::LoadConfig()
03  {
04      char pszTmp[128] = {0};
05      /*读取配置文件中的数据*/
06      GetPrivateProfileString("HERO", "time", "0",
07          pszTmp, 127, ".\\hero.ini");
08      m_uHighTime = atoi(pszTmp);             //转换数据
09
10      m_uXNum     = 30;                      //X 坐标上的方块个数
11      m_uYNum     = 16;                      //Y 坐标上的方块个数
12      m_uMineNum  = 99;                      //地雷个数
13
14      m_bMarkful  = TRUE;
```

```
15      m_bColorful = TRUE;
16  }
17  /*载入图片资源*/
18  void CMyMine::LoadBitmap()
19  {
20      if (m_bColorful) {
21          m_clrDark = COLOR_DARK_GRAY;
22          m_bmpMine.DeleteObject();
23          m_bmpMine.LoadBitmap(IDB_MINE_COLOR);      //地雷图片
24          m_bmpNumber.DeleteObject();
25          m_bmpNumber.LoadBitmap(IDB_NUM_COLOR);     //数字图片
26          m_bmpButton.DeleteObject();
27          m_bmpButton.LoadBitmap(IDB_BTN_COLOR);     //按钮图片
28      }
29      else {
30          m_clrDark = COLOR_BLACK;
31          m_bmpMine.DeleteObject();
32          m_bmpMine.LoadBitmap(IDB_MINE_GRAY);
33          m_bmpNumber.DeleteObject();
34          m_bmpNumber.LoadBitmap(IDB_NUM_GRAY);
35          m_bmpButton.DeleteObject();
36          m_bmpButton.LoadBitmap(IDB_BTN_GRAY);
37      }
38  }
39  /*初始化游戏*/
40  void CMyMine::InitGame()
41  {
42      LoadBitmap();                           //加载图片
43      LoadConfig();                           //加载配置文件
44      m_nLeaveNum = m_uMineNum;
45      m_uSpendTime = 0;
46      m_uBtnState = BS_NORMAL;                //设置当前方块状态
47      m_uGameState = GS_WAIT;                 //设置当前游戏状态
48      if (m_uTimer) {
49          KillTimer(ID_TIMER_EVENT);
50          m_uTimer = 0;
51      }
52      m_pNewMine = NULL;                      //清空当前选中的小方块
53      m_pOldMine = NULL;                      //清空上次选中的小方块
54      //初始化表示雷区的二维数组
55      for (UINT i = 0; i<m_uYNum; i++) {
56          for (UINT j = 0; j<m_uXNum; j++) {
57              m_pMines[i][j].uRow = i;
58              m_pMines[i][j].uCol = j;
59              m_pMines[i][j].uState = STATE_NORMAL;
60              m_pMines[i][j].uAttrib = ATTRIB_EMPTY;
61              m_pMines[i][j].uOldState = STATE_NORMAL;
62          }
63      }
64  }
65
66  void CMyMine::StartGame()                   //开始新游戏接口函数
67  {
68      InitGame();
69      Invalidate();                           //使主界面失效，重新绘制
70  }
```

代码解析：代码第 18 行是实现图片资源的载入，分别载入地雷、数字及按钮图片。

15.5.2　地雷格子模块的设计与实现

地雷格子的处理是扫雷游戏的核心内容，包括如下几个部分。

1. 地雷铺设模块

游戏中的地雷是随机铺设的，可以调用随机数发生函数生成随机数。利用随机数去除最大行数或者最大列数，得到放置地雷行列坐标，然后分别放置地雷到不同的行和列的格子中去。

2. 自动打开周围不是地雷的格子

在游戏中，当玩家单击的格子周围没有地雷格子时，就需要程序自动地把周围的格子自动打开来提高玩家的效率。其实现是通过递归的方法不断地打开当前格子周围地雷个数为 0 的格子来实现。

3. 获得周围地雷个数模块

在游戏中当玩家打开一个格子时，如果当前这个格子不是地雷，那么其一定是标明周围的地雷个数的格子。要实现这个功能主要是通过遍历当前格子周围的 3×3 范围的数组。当找到一个元素状态是地雷时，就把记录增加 1，直到 9 个格子全部找完。这样就可以得到当前格子周围的地雷个数。

以上讲解的相关函数的实现，如代码 15.8 所示。

代码 15.8　地雷格子相关函数的实现

```
01  /*在雷区铺设地雷*/
02  void CMyMine::LayMines(UINT row, UINT col)
03  {
04      srand( (unsigned)time( NULL ) );        //初始化随机数生成种子
05      UINT i, j;
06      for(UINT index = 0; index < m_uMineNum;) {
07          i = rand() % m_uYNum;               //根据生成的随机数得到数组坐标
08          j = rand() % m_uXNum;
09          if (i == row && j == col) continue;
            /*设置地雷*/
10          if(m_pMines[i][j].uAttrib != ATTRIB_MINE) {
11              m_pMines[i][j].uAttrib = ATTRIB_MINE;
12              index++;
13          }
14      }
15  }
16
17  /*自动打开相关不是地雷的格子*/
18  void CMyMine::ExpandMines(UINT row, UINT col)
19  {
20      UINT i, j;
21      UINT minRow = (row == 0) ? 0 : row - 1;
22      UINT maxRow = row + 2;
23      UINT minCol = (col == 0) ? 0 : col - 1;
24      UINT maxCol = col + 2;
```

```
25      UINT around = GetAroundNum(row, col);  //得到周围地雷数量
26
27      m_pMines[row][col].uState = 15 - around;
28      m_pMines[row][col].uOldState = 15 - around;
29      //在指定位置画出地雷
30      DrawSpecialMine(row, col);
31      if (around == 0) {
32          for (i = minRow; i < maxRow; i++) {
33              for (j = minCol; j < maxCol; j++) {
34                  if (!(i == row && j == col) &&
35                      m_pMines[i][j].uState == STATE_NORMAL
36                      && m_pMines[i][j].uAttrib != ATTRIB_MINE) {
37                      if (!IsInMineArea(i, j)) continue;
38                      ExpandMines(i, j);
39                  }
40              }
41          }
42      }
43  }
44  /*获得周围地雷个数*/
45  UINT CMyMine::GetAroundNum(UINT row, UINT col)
46  {
47      UINT i, j;
48      UINT around = 0;                        //初始化变量
49      UINT minRow = (row == 0) ? 0 : row - 1;//得到最小列
50      UINT maxRow = row + 2;
51      UINT minCol = (col == 0) ? 0 : col - 1;//得到最大行
52      UINT maxCol = col + 2;
53
54      for (i = minRow; i < maxRow; i++) {
55          for (j = minCol; j < maxCol; j++) {
56              if (!IsInMineArea(i, j)) continue; //查找指定位置是否是地雷
57              if (m_pMines[i][j].uAttrib == ATTRIB_MINE) around++;
58          }
59      }
60      return around;                              //返回周围地雷个数
61  }
62
63  /*得到周围格子状态*/
64  UINT CMyMine::GetAroundFlags(UINT row, UINT col)
65  {
66      UINT i, j;
67      UINT flags = 0;
68      UINT minRow = (row == 0) ? 0 : row - 1;
69      UINT maxRow = row + 2;
70      UINT minCol = (col == 0) ? 0 : col - 1;
71      UINT maxCol = col + 2;
72
73      for (i = minRow; i < maxRow; i++) {     //遍历指定格子周围的数组数据
74          for (j = minCol; j < maxCol; j++) {
75              if (!IsInMineArea(i, j)) continue;
76              if (m_pMines[i][j].uState == STATE_FLAG) flags++;
77          }
78      }
79      return flags;
80  }
81
82  /*地雷判断*/
83  BOOL CMyMine::IsMine(UINT row, UINT col)
```

```
84  {    /*比较指定数据是否是地雷*/
85       return (m_pMines[row][col].uAttrib == ATTRIB_MINE);
86  }
87
88  /*雷区判断*/
89  BOOL CMyMine::IsInMineArea(UINT row, UINT col)
90  {    /*判断是否是在雷区以内*/
91       return (row >= 0 && row < m_uYNum && col >= 0 && col < m_uXNum);
92  }
```

代码解析：代码第 7 行利用随机数函数生成指定随机数坐标，利用随机数去除最大行数或者最大列数，得到放置地雷行列坐标。然后分别放置地雷到不同的行和列的格子中去。代码第 18 行，地雷展开函数主要是通过递归的方法不断地打开当前格子周围地雷个数为 0 的格子来实现。

15.5.3　游戏规则模块的设计与实现

游戏规则模块的实现，主要由游戏结束和游戏胜利判断函数组成。通过对游戏的结果进行判断，实现扫雷游戏的规则。

游戏规则模块的实现函数，如代码 15.9 所示。

代码 15.9　游戏胜利和失败判断函数的实现

```
01  /*获得胜利*/
02  BOOL CMyMine::Victory()
03  {
04       UINT i, j;
05       CRect rcBtn(m_uBtnRect[1], 15, m_uBtnRect[2], 39);
06
07       for (i = 0; i < m_uYNum; i++) {//遍历整个雷区数组是否还有数据为地雷标志
08           for (j = 0; j < m_uXNum; j++) {
09               if (m_pMines[i][j].uState == STATE_NORMAL) return FALSE;
10               if (m_pMines[i][j].uState == STATE_DICEY) return FALSE;
11           }
12       }
13
14       m_uBtnState = BS_VICTORY;        //设置按钮的状态为胜利
15       m_uGameState = GS_VICTORY;       //设置游戏的状态为胜利
16       Invalidate();
17       if (m_uTimer != 0) {
18           KillTimer(ID_TIMER_EVENT);  //关闭定时器
19           m_uTimer = 0;
20       }
21
22       if(m_uSpendTime < m_uHighTime)  //比较花费时间与记录时间
23       {
24           CHeroDlg dlg;
25
26           dlg.m_time = m_uSpendTime;  //如果小于，就进行记录
27
28           dlg.SetWriteFlg(TRUE);
29
30           dlg.DoModal();              //弹出记录对话框
31       }
```

```
32
33      return TRUE;
34  }
35
36  /*游戏结束*/
37  void CMyMine::Dead(UINT row, UINT col)
38  {
39      CRect rcBtn(m_uBtnRect[1], 15, m_uBtnRect[2], 39);
40      CRect rcMineArea(MINE_AREA_LEFT, MINE_AREA_TOP,
41          MINE_AREA_LEFT + m_uXNum * MINE_WIDTH,
42          MINE_AREA_TOP + m_uYNum * MINE_HEIGHT);
43
44      UINT i, j;
45      if (m_pMines[row][col].uAttrib == ATTRIB_MINE) { //打开了是地雷的格子
46          for (i = 0; i < m_uYNum; i++) {
47              for (j = 0; j < m_uXNum; j++) {
48                  m_pMines[row][col].uState = STATE_BLAST;
49                  m_pMines[row][col].uOldState = STATE_BLAST;
50                  if (m_pMines[i][j].uAttrib == ATTRIB_MINE
51                      && m_pMines[i][j].uState != STATE_FLAG) {
52                      m_pMines[i][j].uState = STATE_MINE;
53                      m_pMines[i][j].uOldState = STATE_MINE;
54                  }
55              }
56          }
57      }
58      else {                                   //打开了判断错误的格子
59          for (i = 0; i < m_uYNum; i++) {
60              for (j = 0; j < m_uXNum; j++) {
61                  m_pMines[row][col].uState = STATE_ERROR;
62                  m_pMines[row][col].uOldState = STATE_ERROR;
63                  if (m_pMines[i][j].uAttrib == ATTRIB_MINE
64                      && m_pMines[i][j].uState != STATE_FLAG) {
65                      m_pMines[i][j].uState = STATE_MINE;
66                      m_pMines[i][j].uOldState = STATE_MINE;
67                  }
68              }
69          }
70      }
71
72      InvalidateRect(rcMineArea);              //重绘指定区域图像
73      m_uBtnState = BS_DEAD;                   //设置按钮状态为结束
74      InvalidateRect(rcBtn);
75      m_uGameState = GS_DEAD;                  //设置游戏状态为结束
76      if (m_uTimer != 0) {
77          KillTimer(ID_TIMER_EVENT);  s        //关闭定时器
78          m_uTimer = 0;
79      }
80
81  }
```

代码解析：代码第 2 行的 Victory()函数，通过遍历整个地雷数组，查找是否全部的地雷格子已经被标示或者找出来。如果已经全部被找出来或者标示出来，那么说明玩家胜利。反之，说明玩家未胜利，需要继续游戏。

代码第 37 行的 Dead()函数在玩家打开一个格子时，就对当前格子进行判断。如果是地雷，说明玩家不幸踩雷，游戏以失败结束。如果不是地雷，就对玩家选中的格子周围的标示进行判断，如果有判断错误的格子，也说明玩家失败，游戏结束。

15.5.4　游戏绘图模块的设计与实现

在扫雷游戏中，通过绘图模块来实现地雷、格子、地雷个数、当前时间及控制按钮等图片和信息的显示。由如下几个函数组成，其代码如代码 15.10 所示。

代码 15.10　绘图模块的函数实现

```
01  void CMyMine::DrawButton(CPaintDC &dc)
02  {
03      CDC cdc;
04      cdc.CreateCompatibleDC(&dc);
05      cdc.SelectObject(m_bmpButton);
06      //把指定范围的图片绘制到界面上
07      dc.StretchBlt(m_uBtnRect[0], 16, 24, 24, &cdc, 0, 24 * m_uBtnState,
08      24, 24, SRCCOPY);
09      //画按钮的 3D 状态
10      dc.Draw3dRect(m_uBtnRect[1], 15, 26, 26, m_clrDark, m_clrDark);
11  }
12  /*画地雷数字*/
13  void CMyMine::DrawNumber(CPaintDC &dc)
14  {
15      CDC cdc;
16      cdc.CreateCompatibleDC(&dc);
17      cdc.SelectObject(m_bmpNumber);
18
19      dc.Draw3dRect(16, 15, 41, 25, m_clrDark, COLOR_WHITE);
20      dc.Draw3dRect(m_uNumRect[0], 15, 41, 25, m_clrDark, COLOR_WHITE);
21      int num;
22      //得到剩余地雷数的个位、十位和百位数
23      num = (m_nLeaveNum < 0) ? 11 : m_nLeaveNum / 100;
24      dc.StretchBlt(17, 16, 13, 23, &cdc, 0, 276 - 23 * (num+1), 13, 23,
        SRCCOPY);
25      num = (m_nLeaveNum < 0) ? -(m_nLeaveNum - num * 100) / 10 :
26      (m_nLeaveNum - num * 100) / 10;
27      dc.StretchBlt(30, 16, 13, 23, &cdc, 0, 276 - 23 * (num+1), 13, 23,
        SRCCOPY);
28      num = (m_nLeaveNum < 0) ? -m_nLeaveNum % 10 : m_nLeaveNum % 10;
29      dc.StretchBlt(43, 16, 13, 23, &cdc, 0, 276 - 23 * (num+1), 13, 23,
        SRCCOPY);
30      //得到当前消耗的时间的个位、十位和百位数
31      num = m_uSpendTime / 100;
32      dc.StretchBlt(m_uNumRect[0], 16, 13, 23, &cdc, 0, 276 - 23 * (num+1),
33          13, 23, SRCCOPY);
34      num = (m_uSpendTime - num * 100) / 10;
35      dc.StretchBlt(m_uNumRect[0] + 13, 16, 13, 23, &cdc,
36          0, 276 - 23 * (num+1), 13, 23, SRCCOPY);
37      num = m_uSpendTime % 10;
38      dc.StretchBlt(m_uNumRect[0] + 26, 16, 13, 23, &cdc,
39          0, 276 - 23 * (num+1), 13, 23, SRCCOPY);
40  }
41  /*画地雷区域*/
42  void CMyMine::DrawMineArea(CPaintDC &dc)
43  {
44      CDC cdc;
45      cdc.CreateCompatibleDC(&dc);
46      cdc.SelectObject(m_bmpMine);
```

```
47
48      for (UINT i = 0; i<m_uYNum; i++) {
49          for (UINT j = 0; j<m_uXNum; j++) {
50      //根据元素类型，绘制不同的图片
51              dc.StretchBlt(MINEAREA_FRAME_X + 16 * j,
52                  MINEAREA_FRAME_Y + 16 * i, 16, 16, &cdc,
53                  0, 16 * m_pMines[i][j].uState, 16, 16, SRCCOPY);
54          }
55      }
56  }
```

代码解析：代码第 1 行绘按钮函数 DrawButton()，主要通过在游戏中不断地得到当前游戏的状态，根据这个状态，在按钮图片中把指定的坐标范围图片绘制到主界面上。

代码第 13 行 DrawNumber()函数实现将剩余地雷个数和消耗时间的数字显示，主要通过一定的算法，得到当前剩余地雷个数和已经消耗时间的个位、十位和百位数字，把相应的数字图片绘制到指定的区域。

代码第 42 行 DrawMineArea()绘雷区函数，是通过遍历当前雷区数组，并根据当前元素所代表的类型不同，将不同的图片绘制到指定位置。

15.5.5　玩家输入模块的设计与实现

在扫雷游戏中，用得最多的就是鼠标的输入。而鼠标输入又分为鼠标左键单击和右键单击处理两种类型。

1．鼠标左键的处理

要实现鼠标左键的处理，需要如下几步操作。

（1）接收玩家在界面上的鼠标左键输入信息。

（2）对当前鼠标的坐标进行判断。

（3）当在按钮区时，调用控制按钮的处理函数。

（4）当在地雷区时，如果当前游戏的状态是等待输入或者正在运行时，就得到当前所选中的格子指针；反之，不进行响应。

（5）在得到格子指针后，对选中格子的状态进行判断。如果是正常的格子，即未被打开过的格子，就转到格子处理函数；如果不是正常的格子，则不进行响应。

2．鼠标右键的处理

实现鼠标右键的处理，需要如下几步操作。

（1）接收玩家在界面上的鼠标右键输入信息。

（2）对当前鼠标的坐标进行判断。

（3）当在地雷区时，如果当前游戏的状态是等待输入或者正在运行时，就得到当前所选中的格子指针；反之，不进行响应。

（4）根据当前格子的状态进行变化。其变化规则为：如果是正常状态，则变成标记（旗子）状态；如果是标记状态，则变成未知（?号）状态；如果是未知状态，则变成正常状态。

鼠标输入模块的函数实现如代码 15.11 所示。

代码 15.11　鼠标输入处理函数实现

```
01  /*鼠标左键输入响应函数*/
02  void CMyMine::OnLButtonDown(UINT nFlags, CPoint point)
03  {
04      CRect rcBtn(m_uBtnRect[1], 15, m_uBtnRect[2], 39);
05      CRect rcMineArea(MINE_AREA_LEFT, MINE_AREA_TOP,
06          MINE_AREA_LEFT + m_uXNum * MINE_WIDTH,
07          MINE_AREA_TOP + m_uYNum * MINE_HEIGHT);
08
09      SetCapture();                                   //捕获鼠标光标
10      m_bClickBtn = FALSE;
11      m_bLRBtnDown = FALSE;
12      if (rcBtn.PtInRect(point)) {                    //单击的按钮区
13          m_bClickBtn = TRUE;
14          m_uBtnState = BS_DOWN;
15          InvalidateRect(rcBtn);
16      }
17      else if (rcMineArea.PtInRect(point)) {  //单击的地雷区
18          switch(m_uGameState)                        //根据当前游戏状态进行处理
19          {
20          case GS_WAIT:                               //游戏状态为等待
21          case GS_RUN:                                //游戏状态为运行
22              m_pNewMine = GetMine(point.x, point.y);
23              if (!m_pNewMine) return;
24              if (m_pNewMine->uState == STATE_NORMAL) {
25                  m_pNewMine->uState = STATE_EMPTY;
26              }
27              if (m_pNewMine->uState == STATE_DICEY) {
28                  m_pNewMine->uState = STATE_DICEY_DOWN;
29              }
30              m_pOldMine = m_pNewMine;
31              break;
32          case GS_DEAD:                               //游戏状态为结束
33          case GS_VICTORY:                            //游戏状态为胜利
34              return;
35          }
36          m_uBtnState = BS_CLICK;                     //按钮为接收按下
37          InvalidateRect(rcBtn);
38          //在左右键同时按下时，进行处理
39          if (nFlags == (MK_LBUTTON | MK_RBUTTON)) {
40              m_bLRBtnDown = TRUE;
41              OnLRBtnDown(m_pOldMine->uRow, m_pOldMine->uCol);
42          }
43          InvalidateRect(rcMineArea);
44      }
45      else {                                          //单击的其他区域
46          if (m_uGameState == GS_WAIT || m_uGameState == GS_RUN) {
47              m_uBtnState = BS_CLICK;
48              InvalidateRect(rcBtn);
49          }
50      }
51
52      CWnd::OnLButtonDown(nFlags, point);     //调用窗口类的左键响应函数
53  }
54  /*鼠标右键输入响应函数*/
55  void CMyMine::OnRButtonDown(UINT nFlags, CPoint point)
56  {
57      CRect rcBtn(m_uBtnRect[1], 15, m_uBtnRect[2], 39);
```

```
58      CRect rcMineArea(MINE_AREA_LEFT, MINE_AREA_TOP,
59          MINE_AREA_LEFT + m_uXNum * MINE_WIDTH,
60          MINE_AREA_TOP + m_uYNum * MINE_HEIGHT);
61
62      m_bLRBtnDown = FALSE;
63      if (rcMineArea.PtInRect(point))         //是单击在地雷区域中
64      {
65          if (m_uGameState == GS_WAIT || m_uGameState == GS_RUN) {
66              m_pNewMine = GetMine(point.x, point.y);
67              if (!m_pNewMine) return;
68              //在左右键同时按下时，进行处理
69              if (nFlags == (MK_LBUTTON | MK_RBUTTON)) {
70                  m_bLRBtnDown = TRUE;
71                  OnLRBtnDown(m_pNewMine->uRow, m_pNewMine->uCol);
72              }
73              else {
74              //根据状态进行格子的变化
75                  switch(m_pNewMine->uState)
76                  {
77                  case STATE_NORMAL:                  //正常状态
78                      m_pNewMine->uState = STATE_FLAG;
79                      m_pNewMine->uOldState = STATE_FLAG;
80                      m_nLeaveNum--;
81                      break;
82                  case STATE_FLAG:                    //标记状态
83                      m_pNewMine->uState = STATE_DICEY;
84                      m_pNewMine->uOldState = STATE_DICEY;
85                      m_nLeaveNum++;
86                      break;
87                  case STATE_DICEY:                   //未知状态
88                      m_pNewMine->uState = STATE_NORMAL;
89                      m_pNewMine->uOldState = STATE_NORMAL;
90                      break;
91                  }
92              }
93              Invalidate();
94          }
95      }
96
97      CWnd::OnRButtonDown(nFlags, point);         //调用窗口类右键处理函数
98  }
```

代码解析：代码第 2 行的 OnLButtonDown()函数是鼠标左键的处理的代码实现。代码第 55 行的 OnRButtonDown()函数，是鼠标右键处理的代码实现。

15.6　扫雷游戏的整合测试

在这一节中，笔者将根据前面的测试用例文档进行扫雷游戏的整合测试。在这里，笔者只对其中几个主要的测试用例进行演示。

15.6.1　主菜单和界面显示功能的测试演示

这个测试用例主要是测试游戏的菜单和界面显示是否成功，根据测试用例文档，其测

试步骤如下所述。

（1）运行扫雷程序，选中其中的.exe 图标，如图 15.7 所示。

（2）程序启动后，其菜单及主界面如图 15.8 所示。

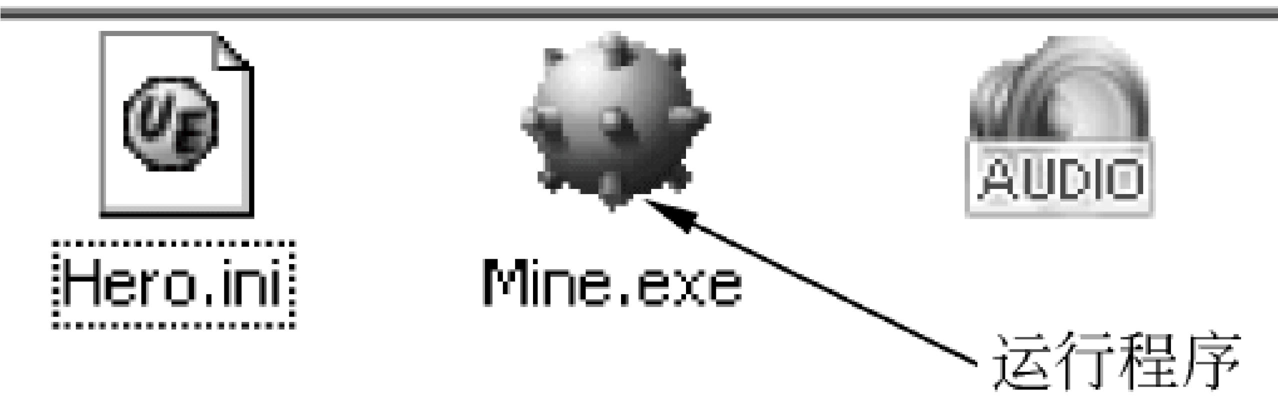


图 15.7　运行扫雷程序

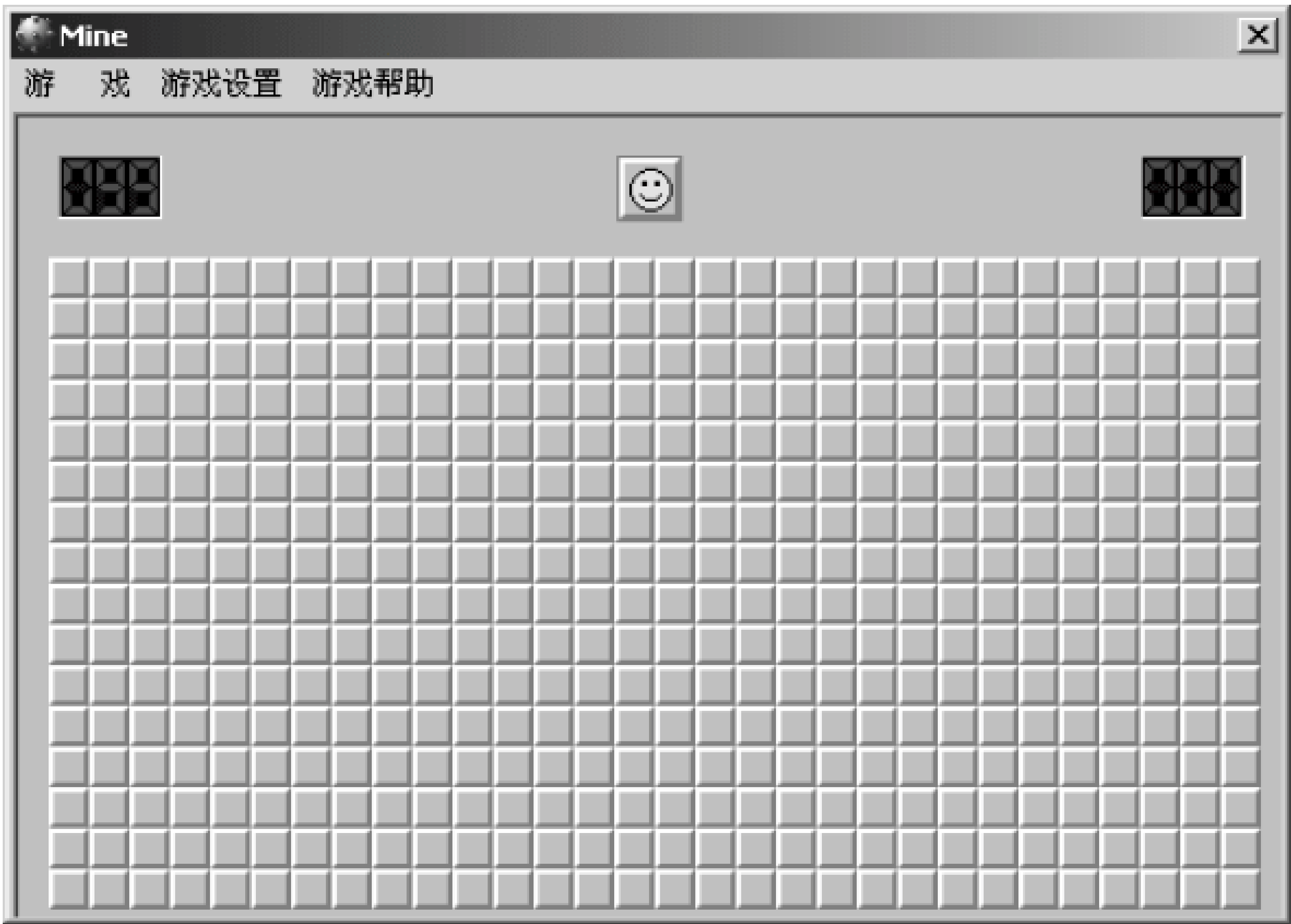


图 15.8　扫雷游戏主界面及菜单

判断结果：游戏的菜单和界面显示成功。填写测试用例编号 1.7.1 结果为“通过”。

15.6.2　鼠标输入功能的测试演示

鼠标输入功能测试，根据测试用例文档，其测试步骤如下所述。

（1）游戏开始后，使用鼠标单击窗口地雷区中的格子，如图 15.9 所示。

（2）使用鼠标右击主游戏窗口中的格子，如图 15.10 所示。

判断结果：鼠标输入功能测试成功。填写测试用例编号 1.7.2 结果为“通过”。

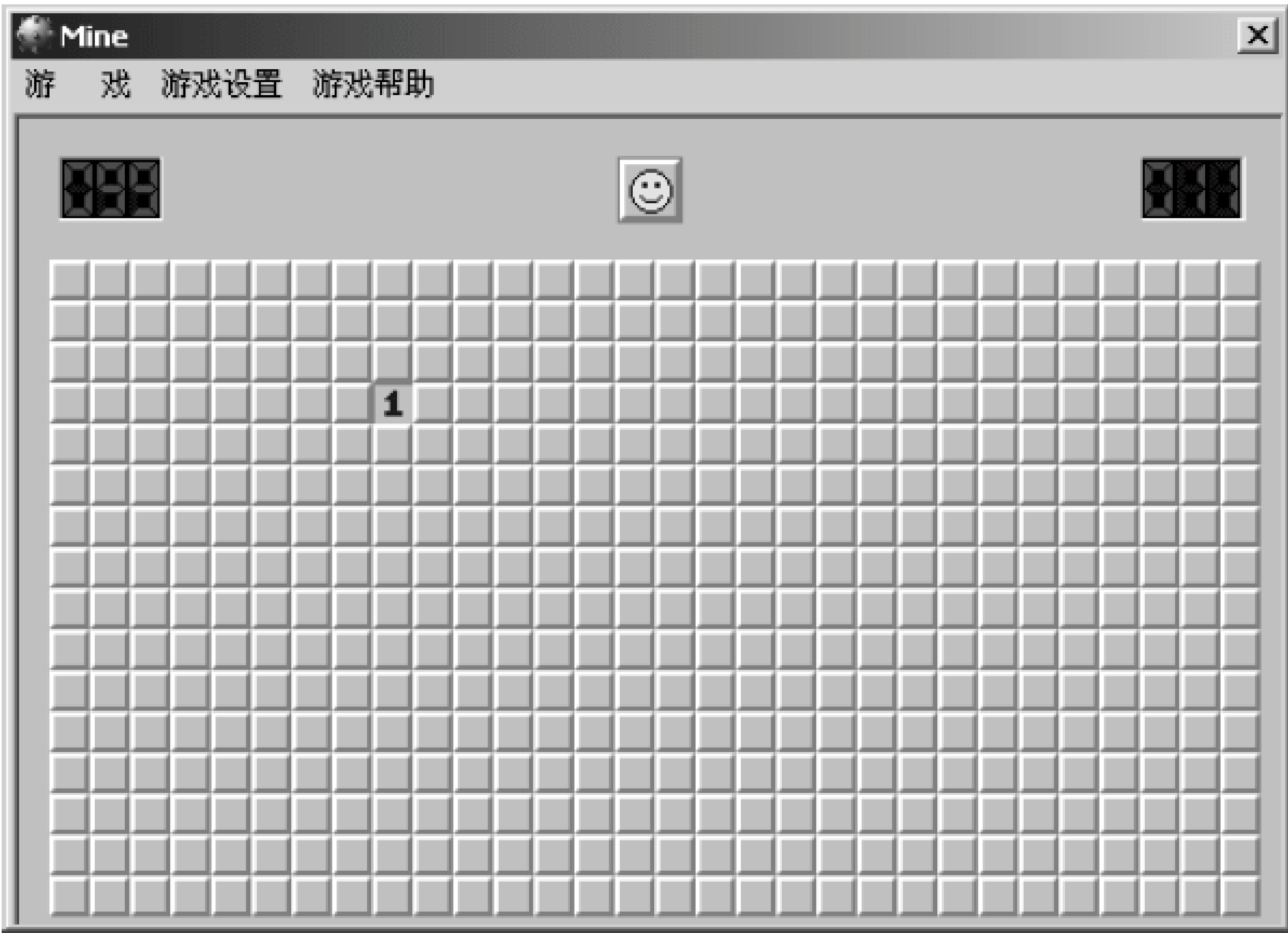


图 15.9　鼠标单击后的格子

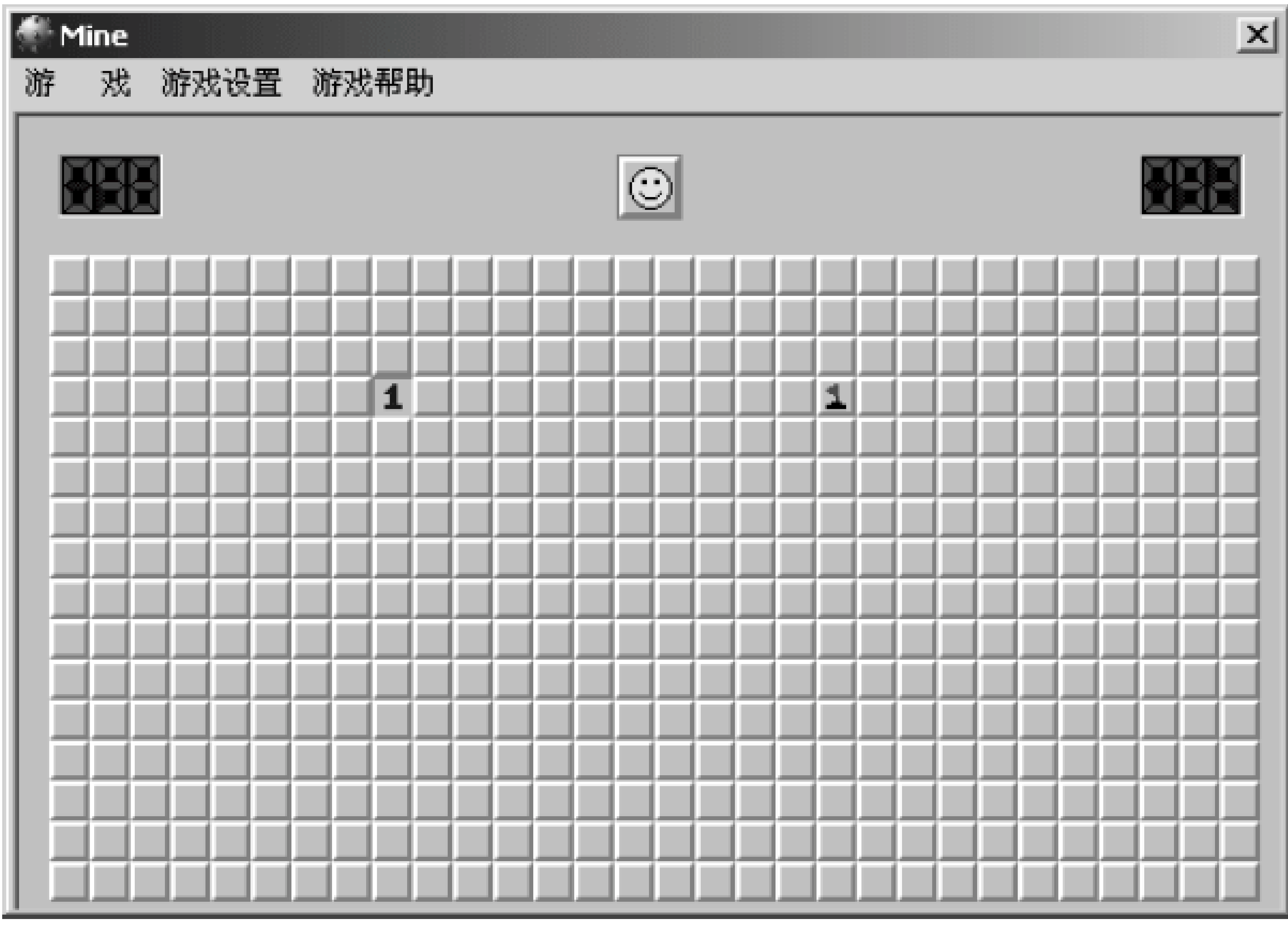


图 15.10　鼠标右击后的格子

15.6.3　标示指定格子功能的测试演示

测试标示指定格子的功能。根据测试用例文档，其测试步骤如下所述。

（1）游戏中格子的初始状态，如图 15.11 所示。

（2）第一次在窗口中指定的多个格子上右击，在相应的格子上面出现的“旗子”标记符号如图 15.12 所示。

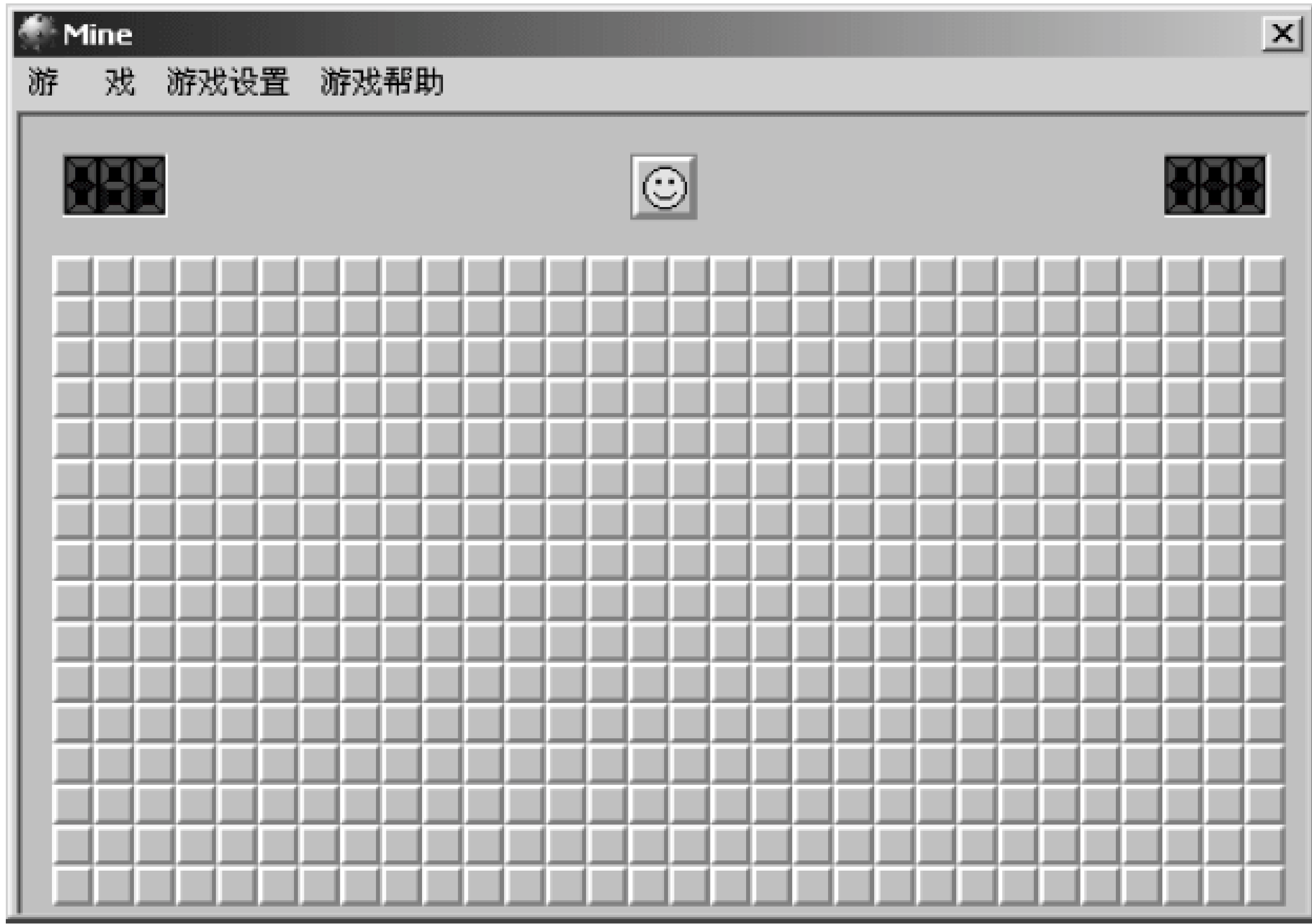


图 15.11　初始状态

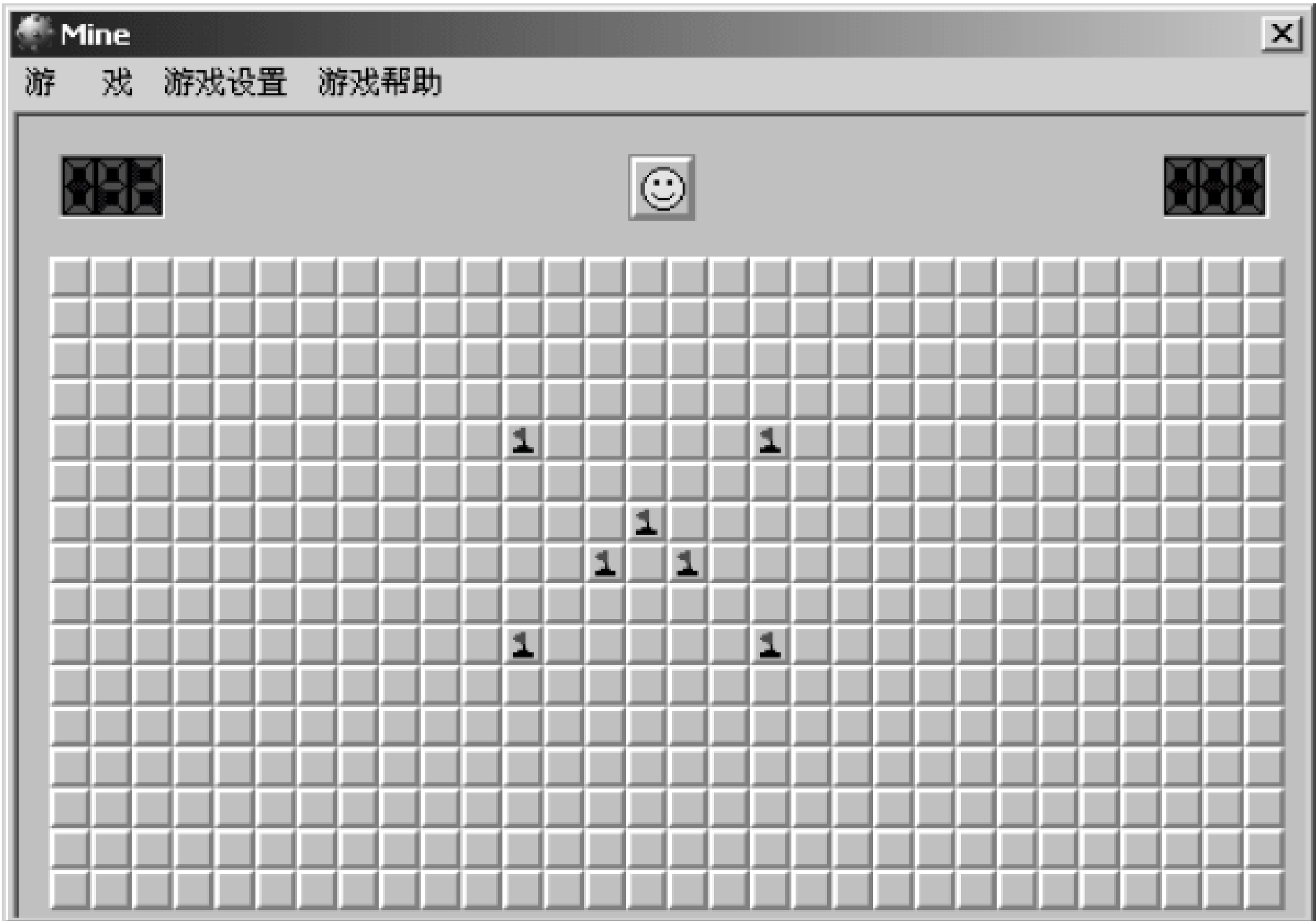


图 15.12　第一次右击后

（3）第二次在窗口中相同的多个格子上右击，在相应的格子上面出现的“?”标记符号如图 15.13 所示。

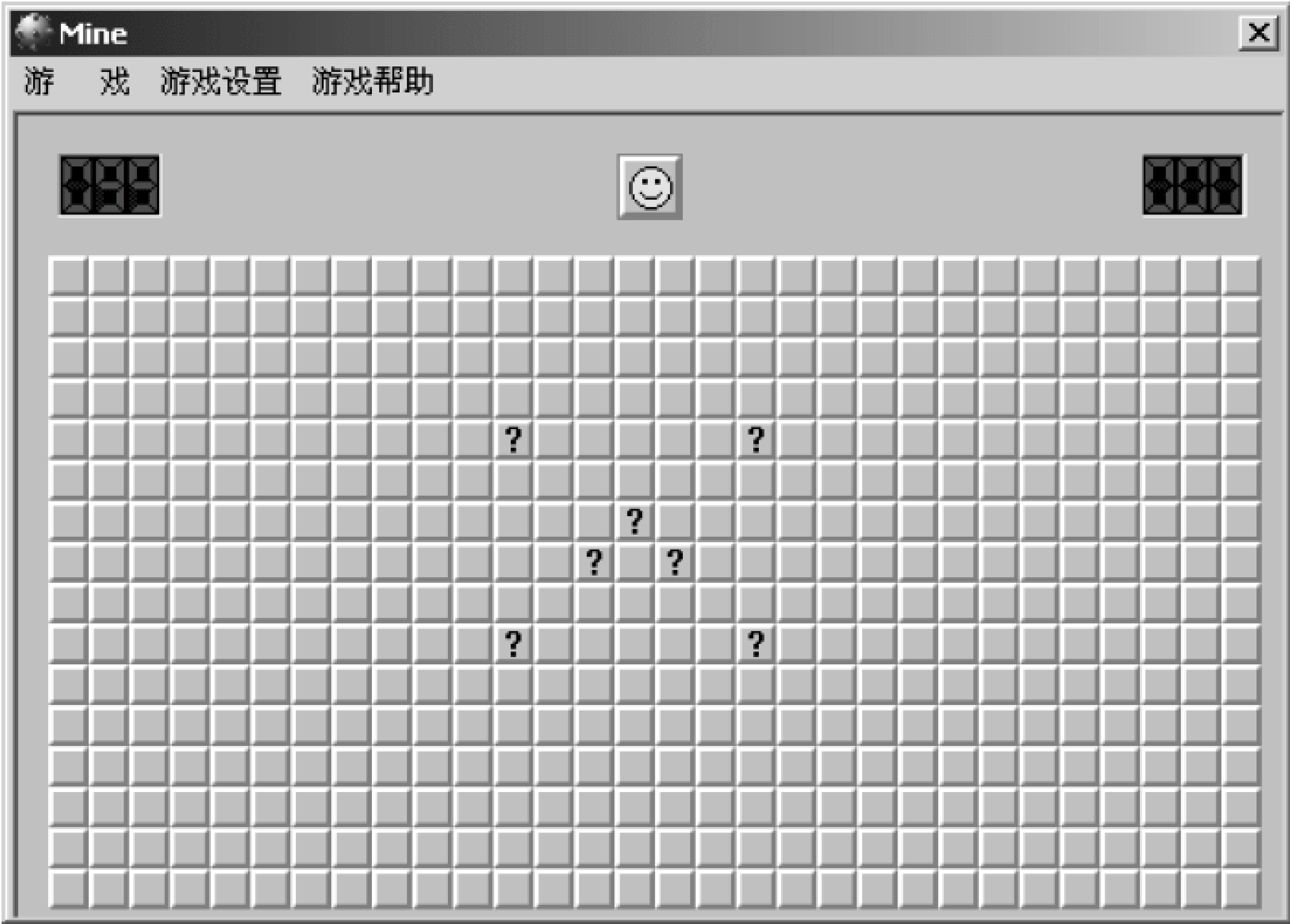


图 15.13　第二次右击后

判断结果：通过比较，标示指定格子的功能测试正确。填写测试用例编号 1.7.3 结果为“通过”。

15.6.4　游戏胜负判断功能的测试演示

测试扫雷游戏中游戏胜负判断功能（这里只对其中的失败状态进行了演示，未对胜利状态进行演示）。根据测试用例文档，其测试步骤如下所述。

（1）游戏开始后，玩家慢慢找出地雷，如图 15.14 所示。

（2）直到有一个地雷格子被不小心踩到，弹出失败提示对话框，如图 15.15 所示。

判断结果：扫雷游戏中胜负判断功能正确。填写测试用例编号 1.7.4 结果为“通过”。

15.6.5　游戏帮助功能的测试演示

测试扫雷游戏是否有帮助提示功能。根据测试用例文档，其测试步骤如下所述。

（1）选择“帮助”|“帮助”命令，如图 15.16 所示。

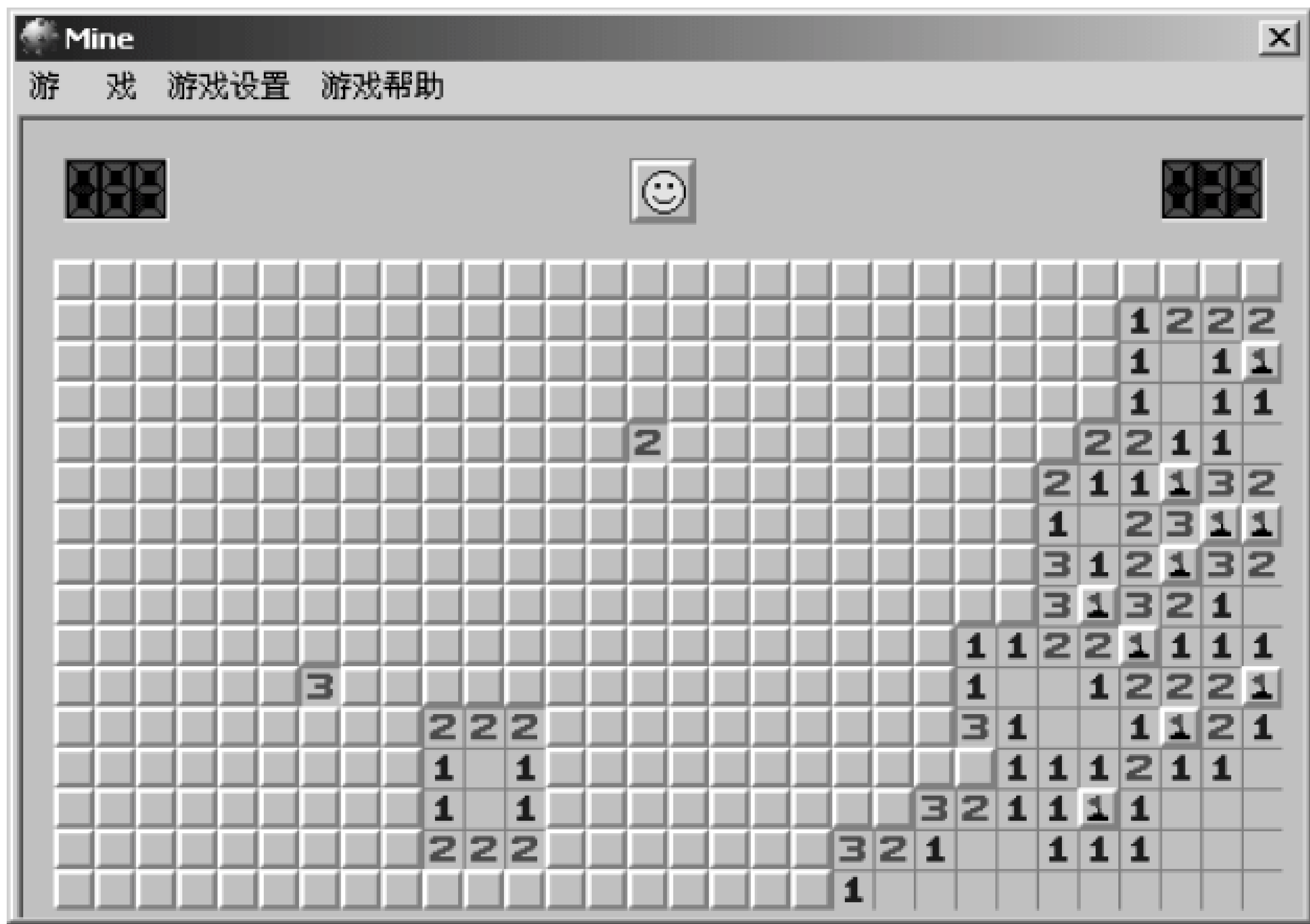


图 15.14　找出地雷



图 15.15　弹出失败提示对话框

（2）游戏中弹出“帮助”对话框，如图 15.17 所示。

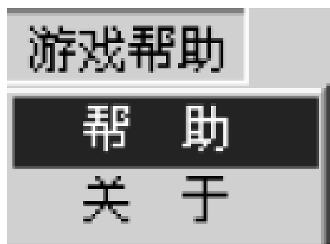


图 15.16　选择“游戏帮助”|“帮助”命令

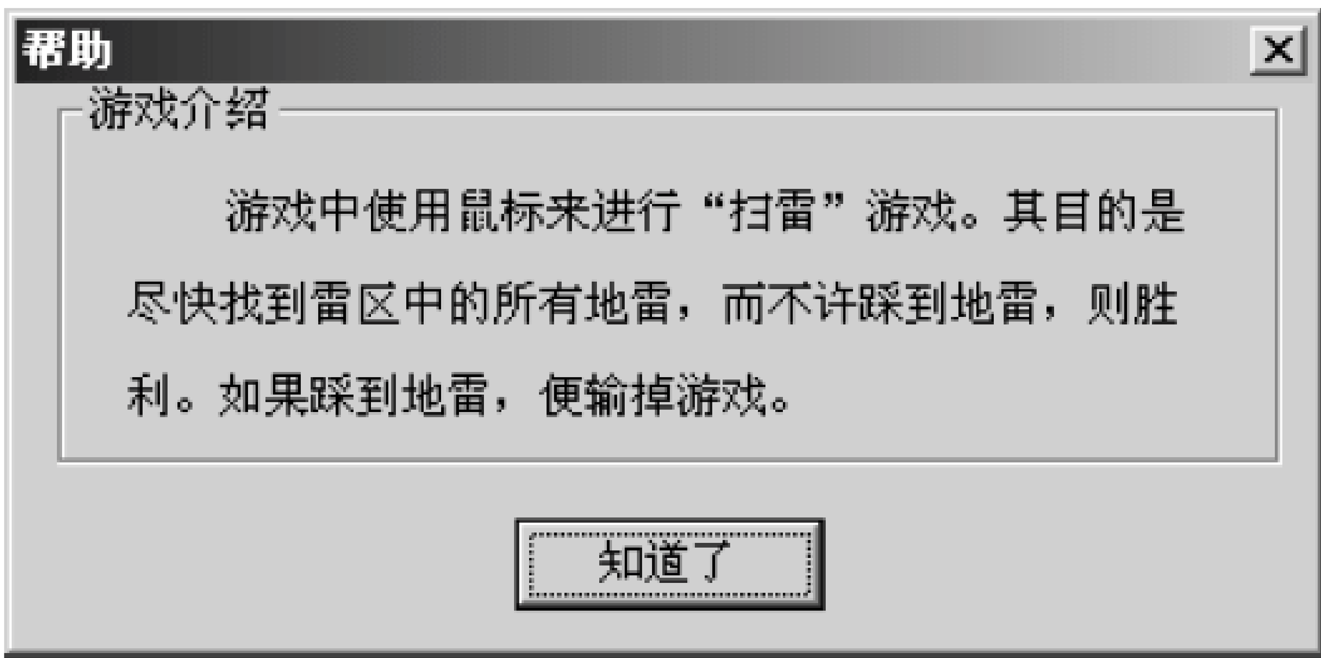


图 15.17　弹出“帮助”对话框

判断结果：扫雷游戏帮助提示是正确的。填写测试用例编号 1.7.6 结果为“通过”。

15.7　总　　结

通过对本章的扫雷游戏实例项目开发的学习，希望读者能够掌握游戏项目开发中各种文档的格式及编写方法。同时，了解扫雷游戏中包括绘图模块、鼠标输入模块、格子模块等核心算法的实现，最终能够自己开发一款扫雷游戏。

实践是学习程序开发最好的方式，所以请读者抽出时间边学习，边实践。

第 16 章　推箱子游戏项目开发

学习完第 15 章的扫雷游戏的开发后，在本书的最后一章笔者将介绍使用游戏开发技术较多的一款“推箱子”游戏的实例项目。主要目的还是为了帮助读者继续增加项目实际开发的经验，提高对各种文档的编写能力。

本章主要涉及的内容如下：

- 推箱子项目的需求分析。
- 推箱子游戏的概要设计。
- 推箱子游戏操作界面设计。
- 推箱子游戏的详细设计及代码。
- 推箱子游戏的测试用例文档的编写及测试演示。

16.1　推箱子游戏项目的需求分析

项目都是由用户提出需求，再由分析师对其进行分析，最后得出需求分析文档。根据这份文档，项目实施人员才能正确地完成项目的目标。

技巧： 用户的需求在时间上总是不断地变化的，为了达到项目的目标，需要根据其需求的变化进行软件版本的规划和升级。

16.1.1　获得客户需求的语言描述

通过与某公司用户的沟通，笔者得到推箱子游戏开发的资料如下所述。

1．推箱子游戏概述

经典的推箱子是一个来自日本的古老游戏，目的是训练玩家的逻辑思考能力。在一个狭小的仓库中，要求把木箱放到指定的位置，稍不小心就会出现箱子无法移动或者通道被堵住的情况，所以需要巧妙地利用有限的空间和通道，合理安排移动的次序和位置，才能顺利地完成任务。

2．推箱子的操作方法

在游戏主界面中，会出现一个小人、若干个箱子和箱子放置点。玩家需要利用方向键控制小人上下左右移动，并推动界面中的箱子到达指定的箱子放置点。

3．推箱子游戏的基本规则

在游戏中，当玩家把全部的箱子都推到箱子放置点时，玩家胜利通过当前游戏关口，则进行下一关口的游戏。如果玩家无法将指定的箱子全部推到放置点时，玩家失败。玩家可以选择重新进行当前关口的游戏，或者退出游戏。

4．新地图扩展功能

玩家通过编辑地图文件，能够支持自己扩展新关口地图的功能。

5．玩家可以自由选择当前游戏关口

玩家可以选择自己想要挑战或者重新玩的关口。

6．有背景音乐支持

在游戏中，能够通过选择播放背景音乐。

7．游戏的帮助

在游戏界面中需要提供游戏使用说明等帮助提示，以方便对本游戏不了解的玩家对游戏进行操作和使用。

16.1.2　对语言描述进行需求分析

根据《推箱子用户需求描述文档》中的内容，现在需要对其进行需求分析，将其转换为程序员能阅读的项目需求文档。

技巧：在需求分析文档中，应当把用户提出的每个功能分别列出，并加以详细说明。这样在做项目计划时，方便进行时间结点的工作安排。

1．引言

某公司为了扩大公司的知名度，需要开发一款单机版的休闲类推箱子游戏，特制定本说明书来用于描述某公司推箱子项目开发的功能性需求。

1.1　编写目的

使用技术性语言对某公司的推箱子游戏项目开发的需求进行描述。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．文档范围

包含某公司推箱子游戏项目的开发需求。

3．使用对象

本说明书使用对象主要是与某公司推箱子游戏开发相关的需求分析、程序设计、代码编写、测试和维护等部门（单位）的人员。

4．参考文献

《推箱子用户需求描述文档》。

5．游戏具有的功能

5.1　能够显示主菜单和界面

游戏需要提供主菜单让玩家进行游戏设置，同时能够把地图文件中的信息转换成为图像显示到主游戏界面上。

5.2　能够实现键盘操作功能

能够接收到键盘输入的方向键信息，并根据不同的方向键把游戏人物移动到相应的位置。例如，当玩家单击方向键“上”时，如果向上的位置是可移动的，那么就当把游戏人物向上移动一个方格。

5.3　能够把放置到位置上的箱子进行变色显示

当玩家把箱子推到指定位置的格子时，需要把这个箱子进行变色。这样就能明确地显示出该箱子已经放置到指定位置上。

5.4　支持地图扩展功能

玩家可以自己扩展原游戏地图文件，从而生成新的游戏地图。

5.5　游戏胜负判断功能

在游戏中，当玩家把全部的箱子都推到箱子放置点时，玩家胜利通过当前游戏关口，则进行下一关口的游戏。如果玩家无法将指定的箱子全部推到放置点时，玩家失败。玩家可以选择重新进行当前关口的游戏，还是退出游戏。

5.6　支持关口选择功能

玩家在游戏中可自行选择需要挑战的关口。

5.7　游戏支持背景音乐功能

通过主菜单，在游戏开始后，可以选择播放或者禁止播放背景音乐。默认为禁止播放。

5.8　游戏提供帮助说明

在游戏菜单中，提供一个使用说明项，以方便对本游戏不了解的玩家对游戏进行操作和使用。

16.2　推箱子游戏概要设计

笔者编写的推箱子游戏的概要设计文档如下所述。

1．引言

1.1　编写目的

为了让每个开发人员明白推箱子游戏项目的总体设计思路，并且能够按照概要设计的要求完成各功能目标，特制定本文档。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．术语

3．参考文献

《推箱子游戏需求分析说明书》。

4．任务概述

4.1　目标

通过系统分析并与某公司测试人员再次探讨，最终确定游戏的目标如下：

- 实现需求分析阶段客户提出的全部功能。
- 提高鼠标及键盘操作的易用性。

4.2　开发软件及硬件环境

- Intel® Pentium® 4 2.0GHz，512M 内存，80G 硬盘。
- Microsoft® Windows™ 2000 Professional。
- Microsoft® Visual C++ 6.0。

4.3　需求概述

参见《推箱子游戏需求分析说明书》。

4.4　条件与限制

无。

5．总体设计

5.1　推箱子游戏的功能架构（如图 16.1 所示）

5.2　各功能处理流程

参见《推箱子游戏各功能详细设计文档》。

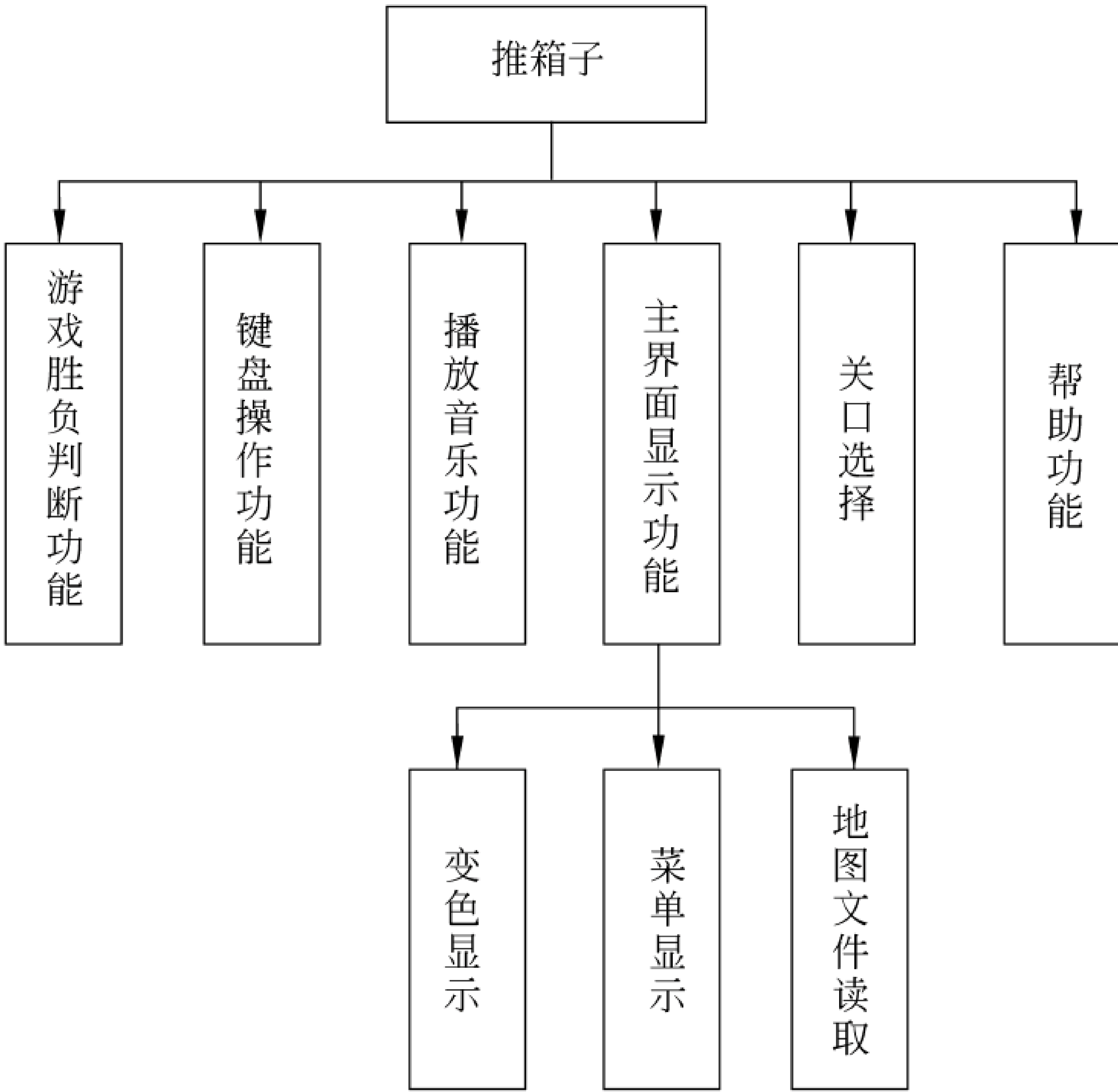


图 16.1　推箱子功能架构

6．接口设计

内容参见《推箱子游戏操作界面设计文档》。

7．程序结构设计

游戏共由 3 个类和 5 个模块组成，如图 16.2 所示。



图 16.2　游戏主要类结构

- 主界面对话框类：主要负责主界面、菜单及各个窗口类对象的创建和调用等处理。
- 键盘操作模块：主要负责接收玩家键盘输入并进行箱子移动等处理。
- 关口选择对话框类：主要负责游戏挑战关口的选择和设置。
- 地图文件读取模块：主要负责读取地图文件并进行相应的解析工作。
- 地图绘制模块：主要负责将地图数组中的数据绘制成地图图像。
- 游戏规则模块：主要负责游戏规则的判断。
- 背景音乐播放模块：主要负责游戏中背景音乐的播放。
- 帮助对话框类：主要负责帮助提示的显示及其他辅助信息。

8．出错处理设计

8.1　出错输出信息

当游戏中出现错误，采用弹出对话框的方式提示用户出现错误。

8.2　出错处理对策

当游戏中出现错误，采用中止当前游戏并重新开始新游戏的方法处理游戏中的错误。

9．维护设计

由于整个推箱子游戏项目在开发完成后，基本不会有太多的变动，所以维护的主要任务是把用户使用中出现的问题解决。

16.3　推箱子游戏操作界面及测试用例设计

游戏操作界面设计文档是用户了解游戏整体界面最直观的方法，通过本文档可以让用户在开始游戏设计前就知道最后的游戏界面是怎样的。本节将继续介绍《游戏操作界面设计文档》和《游戏测试用例文档》的编写。

注意：在游戏操作界面设计文档中，应该突出游戏的主体框架和界面，忽略一些图像细节，这样才能让游戏开发者有更多的自由空间来发挥。

16.3.1　游戏操作界面设计文档

根据前面的需求分析和概要设计文档，笔者编写的推箱子游戏的操作界面设计文档内容如下所述。

1．引言

1.1　编写目的

为了让所有的项目开发人员明确推箱子游戏的操作界面是如何设计的，特制定本文档来用于描述本公司推箱子游戏项目的游戏操作界面。

1.2　项目背景

- 项目提出者：某公司。
- 项目开发者：某软件公司。
- 游戏用户：某公司的测试人员及客户。

2．文档范围

包含本公司推箱子游戏的操作界面设计。

3．使用对象

本说明书使用对象主要是程序设计、代码编写、测试及维护等部门（单位）的人员。

4．参考文献

《推箱子游戏需求分析说明书》；

《推箱子游戏概要设计文档》。

5．游戏界面设计

5.1　游戏主界面的设计

推箱子的游戏主界面设计如图 16.3 所示。

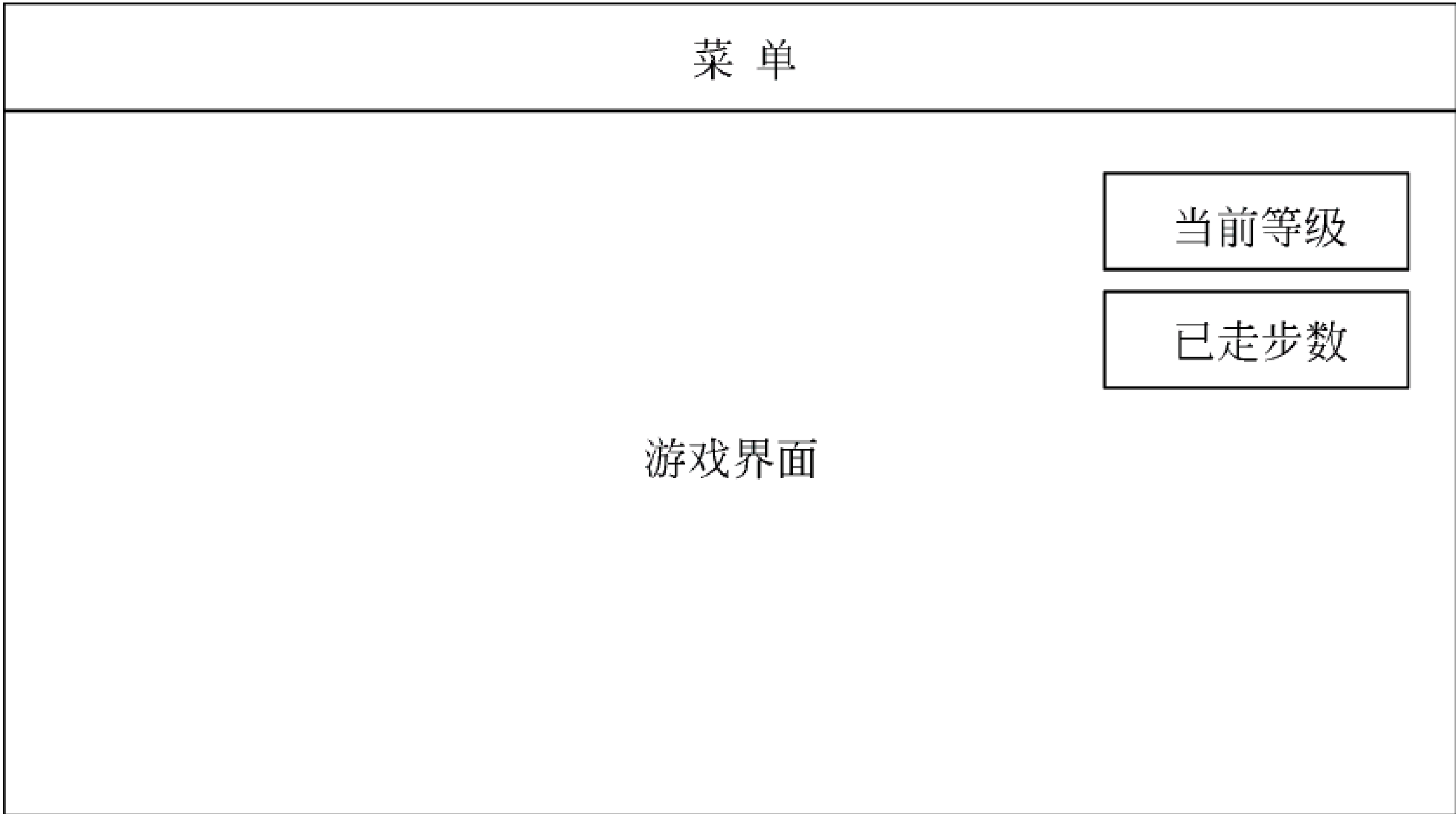


图 16.3　设计的游戏主界面

5.2　游戏菜单结构的设计

推箱子的游戏菜单设计如图 16.4 所示。

5.3　游戏中关口选择对话框的设计

关口选择对话框的设计如图 16.5 所示。

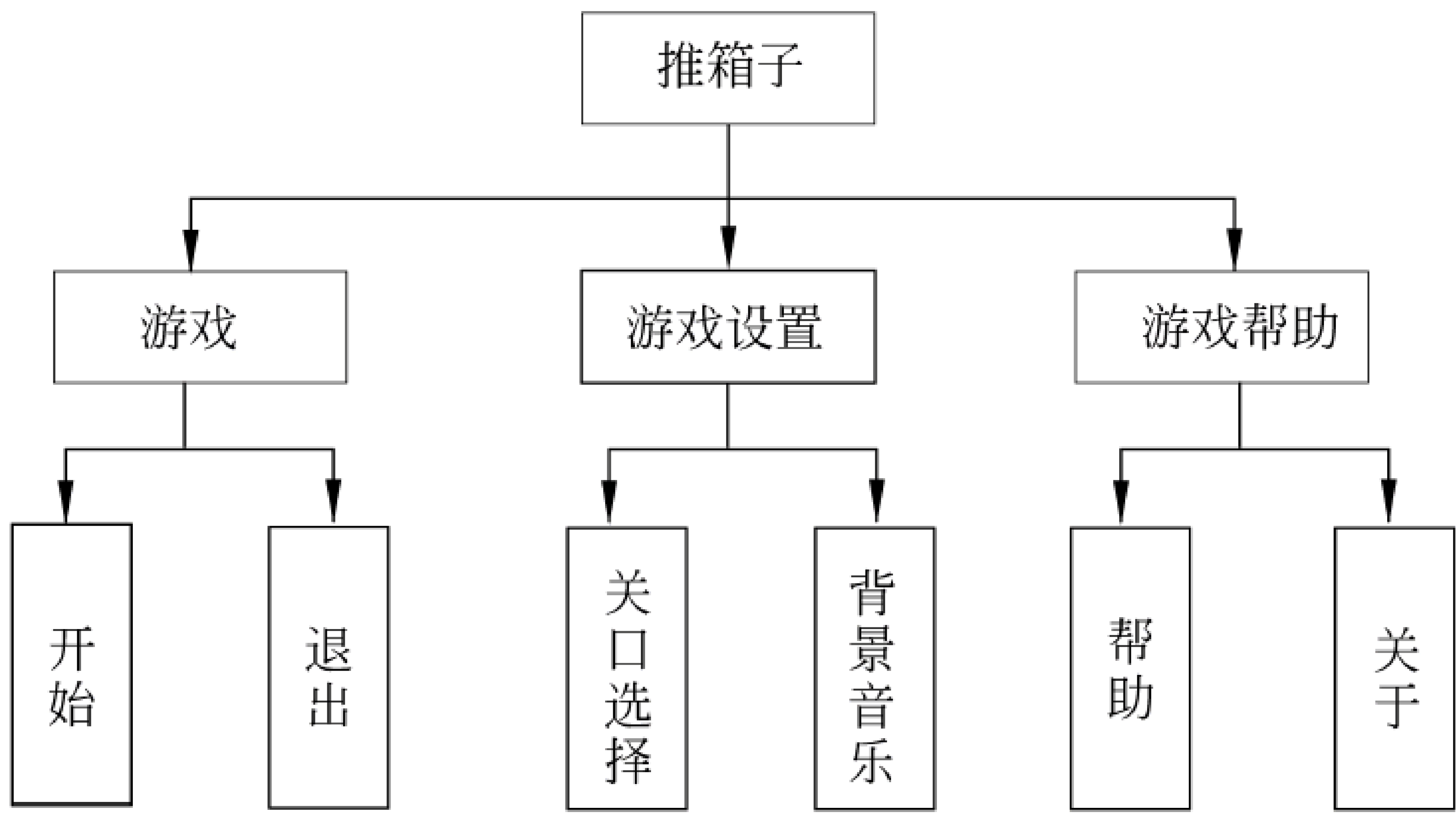


图 16.4　设计的游戏菜单结构



图 16.5　“关口选择”对话框

16.3.2　测试用例文档

测试用例文档主要用于指导测试人员对游戏的各个功能进行测试。笔者编写的推箱子游戏的测试用例文档内容如下所述。

1．引言

本文档主要用于某公司在开展推箱子游戏项目测试时，提供功能测试的实用案例及测试方法说明。

本文档规定了推箱子游戏项目测试中所用到的测试环境和测试方法，主要包括测试环境的配置、测试方法的使用和测试项目等内容。

本文档中的测试大项分为“必测”和“选测”两种。“必测”项又分为 A、B、C 这 3 类，“选测”项为可选部分。只有如下标准满足时，才认为该功能通过测试。

A 类测试项都为“必测”项目。其项目中的内容必须全部通过，方能认定测试合格，符合用户需求。B 类不通过测试项数少于 3 项（含 3 项）；C 类测试项不合格数少于 6 项（含 6 项）。“选测”项的测试结果不对该系统测试总体结论起决定性影响。

本文档由本公司负责解释。

2．文档范围

本测试用例文档对某公司的推箱子游戏项目的测试内容和测试方法提出规定。原则上只能在本公司内部使用，用于指导本公司的测试人员，进行推箱子游戏项目的测试和验收。

3．使用对象

本测试用例文档使用对象主要是与某公司推箱子游戏开发相关的需求分析、测试和维

护等部门（单位）的人员。

4．参考文献

《推箱子游戏的需求分析说明书》；

《推箱子游戏的概要设计文档》；

《推箱子游戏的详细设计文档》。

5．相关术语与缩略语解释

无。

6．测试项目

测试项目主要针对推箱子中的各种功能进行整合性测试，共包含如下几个项目。

（1）主菜单和界面显示功能的测试，主要内容如表 16.1 所示。

表 16.1　主菜单和界面显示功能的测试

测试编号：1.7.1	类别：A
项　　目：推箱子测试	
分 项 目：主菜单和界面显示功能的测试	
测试目的：测试推箱子游戏中的菜单和界面是否正确显示	
测试配置：	
预置条件： 推箱子游戏源程序已经编译完成，并可以运行； 键盘和鼠标已准备好	
测试步骤： 运行推箱子程序，查看菜单和界面	
预期结果： 游戏主界面及菜单与操作设计文档中的一致	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏主界面和菜单是否正确显示（是/否）	
测试结果： 通过/不通过	

（2）键盘操作功能的测试，主要内容如表 16.2 所示。

表 16.2　键盘操作功能的测试

测试编号：1.7.2	类别：A
项　　目：推箱子测试	
分 项 目：键盘操作功能的测试	
测试目的：测试游戏是否支持键盘的操作功能	
测试配置：	
预置条件： 推箱子游戏已经开始； 键盘已经准备好； 主角预移动的方向没有阻碍	

续表

测试步骤： 单击键盘上的“左”方向按键，查看主角是否向左移动； 单击键盘上的“右”方向按键，查看主角是否向右移动； 单击键盘上的“上”方向按键，查看主角是否向上移动； 单击键盘上的“下”方向按键，查看主角是否向下移动
预期结果： 单击键盘上的“左”方向按键时，主角向左移动； 单击键盘上的“右”方向按键时，主角向右移动； 单击键盘上的“上”方向按键时，主角向上移动； 单击键盘上的“下”方向按键时，主角向下移动
判定原则： 测试结果必须与预期结果相符，否则不符合要求
测试记录： 游戏是否支持键盘操作功能（是/否）
测试结果： 通过/不通过

（3）箱子放置到指定位置时进行变色显示功能的测试，主要内容如表 16.3 所示。

表 16.3　箱子放置到指定位置时进行变色显示功能的测试

测试编号：1.7.3	类别：A
项　　目：推箱子测试	
分 项 目：箱子放置到指定位置时进行变色显示功能的测试	
测试目的：测试是否能够对箱子放置到指定位置时进行变色显示	
测试配置：	
预置条件： 推箱子游戏已经开始； 键盘已经准备好	
测试步骤： 移动主角把箱子推到指定放置位置	
预期结果： 当移动箱子到指定位置上时，箱子变成其他颜色显示	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏中是否能够对箱子放置到指定位置时进行变色显示（是/否）	
测试结果： 通过/不通过	

（4）支持地图扩展功能的测试，主要内容如表 16.4 所示。

表 16.4　地图扩展功能的测试

测试编号：1.7.4	类别：A
项　　目：推箱子测试	
分 项 目：地图扩展功能的测试	
测试目的：测试游戏能否支持地图扩展	
测试配置：	
预置条件： 查看第一关地图	
测试步骤： 根据规则地图文件中的第一关进行编辑； 开始游戏	
预期结果： 地图显示与地图文件中的内容一样	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否支持地图扩展（是/否）	
测试结果： 通过/不通过	

（5）游戏胜负判断功能的测试，主要内容如表 16.5 所示。

表 16.5　游戏胜负判断功能的测试

测试编号：1.7.5	类别：A
项　　目：推箱子测试	
分 项 目：游戏胜负判断功能的测试	
测试目的：测试推箱子游戏能否对游戏胜负进行判断	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 玩家一步步把全部的箱子推到指定放置位置； 玩家把箱子错误地推到不是指定的放置位置并不能再移动，导致游戏无法进行，玩家退出	
预期结果： 玩家获得胜利； 玩家选择退出，提示失败	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否对游戏胜负进行判断（是/否）	
测试结果： 通过/不通过	

（6）支持关口选择功能的测试，主要内容如表 16.6 所示。

表 16.6　支持关口选择功能的测试

测试编号：1.7.6	类别：A
项　　目：推箱子测试	
分 项 目：支持关口选择功能的测试	
测试目的：测试推箱子游戏能够进行关口的选择	
测试配置：	
预置条件： 游戏已经运行； 鼠标和键盘都准备好	
测试步骤： 选中菜单中的“游戏设置”|“关口选择”菜单栏； 在弹出的对话框中设置需要挑战的关口	
预期结果： 从设置好的挑战的关口开始游戏	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否进行关口的选择（是/否）	
测试结果： 通过/不通过	

（7）背景音乐播放功能的测试，主要内容如表 16.7 所示。

表 16.7　背景音乐播放功能的测试

测试编号：1.7.7	类别：A
项　　目：推箱子测试	
分 项 目：背景音乐播放功能的测试	
测试目的：测试推箱子游戏能否支持播放背景音乐	
测试配置：	
预置条件： 游戏已经运行	
测试步骤： 选中“游戏设置”|“背景音乐”菜单栏	
预期结果： 通过喇叭能够听到有背景音乐声响起	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 游戏能否支持播放背景音乐（是/否）	
测试结果： 通过/不通过	

（8）帮助功能的测试，主要内容如表 16.8 所示。

表 16.8　帮助功能的测试

测试编号：1.7.8	类别：A
项　　目：推箱子测试	
分 项 目：帮助功能的测试	
测试目的：测试推箱子游戏是否有帮助提示功能	
测试配置：	
预置条件： 鼠标已经准备好； 游戏已经可以运行	
测试步骤： 选择“游戏帮助”|“帮助”命令	
预期结果： 出现游戏帮助提示，说明游戏操作方法	
判定原则： 测试结果必须与预期结果相符，否则不符合要求	
测试记录： 推箱子游戏是否有帮助提示功能（是/否）	
测试结果： 通过/不通过	

16.4　推箱子游戏的界面实现

推箱子游戏的 Visual C++工程采用 MFC 对话框模式进行开发。本节主要讲解推箱子游戏各个功能模块的代码实现。

技巧：用户界面如果是自己设计的，那么其在不同的操作系统中，表现出来的效果都一样，这样大大地方便了用户的使用。

16.4.1　游戏菜单的实现

在推箱子游戏中，通过如下几步即可实现游戏的菜单。

（1）在推箱子游戏工程的资源中添加一个菜单资源，其属性如表 16.9 所示。

表 16.9　主菜单属性

ID	类　　别	说　　明
IDR_MAIN_MENU	弹出菜单	游戏的主菜单
IDR_START_GAME	菜单栏	开始游戏
IDR_EXIT_GAME	菜单栏	退出游戏

续表

ID	类　别	说　明
IDR_SELECT_LEVEL	菜单栏	关口选择
IDR_PLAY_MUSIC	选择菜单	播放音乐
IDR_HELP	菜单栏	帮助
IDR_ABOUT	菜单栏	关于

（2）给每个菜单栏添加响应函数到 CBoxManDlg 类中。

（3）菜单响应函数的实现，如代码 16.1 所示。

代码 16.1　菜单响应函数的实现

```
01  …                                  //省略部分代码
02  BEGIN_MESSAGE_MAP(CBoxManDlg, CDialog)
03      /*函数与菜单资源映射表*/
04      ON_COMMAND(IDR_EXIT_GAME, OnExitGame)
05      ON_COMMAND(IDR_HELP, OnHelp)
06      ON_COMMAND(IDR_PLAY_MUSIC, OnPlayMusic)
07      ON_COMMAND(IDR_SELECT_LEVEL, OnSelectLevel)
08      ON_COMMAND(IDR_START_GAME, OnStartGame)
09      ON_COMMAND(IDR_ABOUT, OnAbout)
10  END_MESSAGE_MAP()
11  ...                                //省略部分代码
12  void CBoxManDlg::OnOK()            //确认响应函数
13  {
14      CDialog::OnOK();
15  }
16
17  void CBoxManDlg::OnCancel()        //退出响应函数
18  {
19      CDialog::OnCancel();
20  }
21
22  BOOL CBoxManDlg::OnInitDialog()
23  {
24      CDialog::OnInitDialog();
25
26      SetIcon(m_hIcon, TRUE);        //设置大图标
27      SetIcon(m_hIcon, FALSE);       //设置小图标
28
29      m_bStart = FALSE;              //设置游戏状态
30
31      InitMenu();                    //初始化菜单
32
33      return TRUE;                   //初始化成功
34  }
35
36  void CBoxManDlg::OnExitGame()      //退出菜单栏响应函数
37  {
38      CDialog::OnCancel();           //调用基类退出函数
39  }
40  void CBoxManDlg::OnHelp()          //帮助菜单栏响应函数
41  {
42      CHelpDlg dlg;                  //创建帮助对话框对象
43      dlg.DoModal();                 //弹出对话框
```

```
44  }
45
46  void CBoxManDlg::OnPlayMusic()      //背景音乐菜单栏响应函数
47  {
48      CWnd*  pMain   =  AfxGetMainWnd();
49      CMenu*  pMenu   =  pMain->GetMenu();
50      //判断播放音乐菜单当前状态
51      BOOL bCheck = (BOOL)pMenu->GetMenuState(IDR_PLAY_MUSIC,
        MF_CHECKED);
52
53      if(m_bStart)                    //游戏是否开始判断
54      {
55          if(bCheck)
56          {
57              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
58                  MF_BYCOMMAND | MF_UNCHECKED);
59          }
60          else
61          {
62              pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
63                  MF_BYCOMMAND | MF_CHECKED);
64          }
65
66          PlayBackMusic(!bCheck);         //调用播放背景音乐功能函数
67      }
68  }
69
70  void CBoxManDlg::OnSelectLevel()        //关口选择菜单栏响应函数
71  {
72      CSelectDlg dlg;                     //创建关口选择对话框对象
73      dlg.DoModal();                      //弹出关口选择对话框
74  }
75
76  void CBoxManDlg::OnStartGame()          //开始游戏菜单栏响应函数
77  {
78      GameStart();                        //调用开始游戏成员函数
79  }
80
81  void CBoxManDlg::OnAbout()              //关于菜单栏响应函数
82  {
83      CAboutDlg dlg;                      //创建关于对话框对象
84      dlg.DoModal();                      //弹出关于对话框
85  }
```

代码解析：代码第 31 行是调用初始化菜单函数进行菜单格式的初始化。代码第 78 行是开始游戏菜单响应函数的实现，主要是通过调用游戏开始函数进行游戏。

16.4.2　游戏帮助对话框的实现

推箱子游戏中的帮助是使用一个对话框来实现的。其实现步骤如下所述。

（1）添加一个对话框资源到工程中，并填写说明文字，如图 16.6 所示。

（2）编写一个 CHelpDlg 对话框类，主要是加载 IDD_HELP 对话框资源，通过资源中的文字说明对游戏操作方法进行描述。同时只包含单击“知道了”按钮的响应函数。其类声明如代码 16.2 所示。

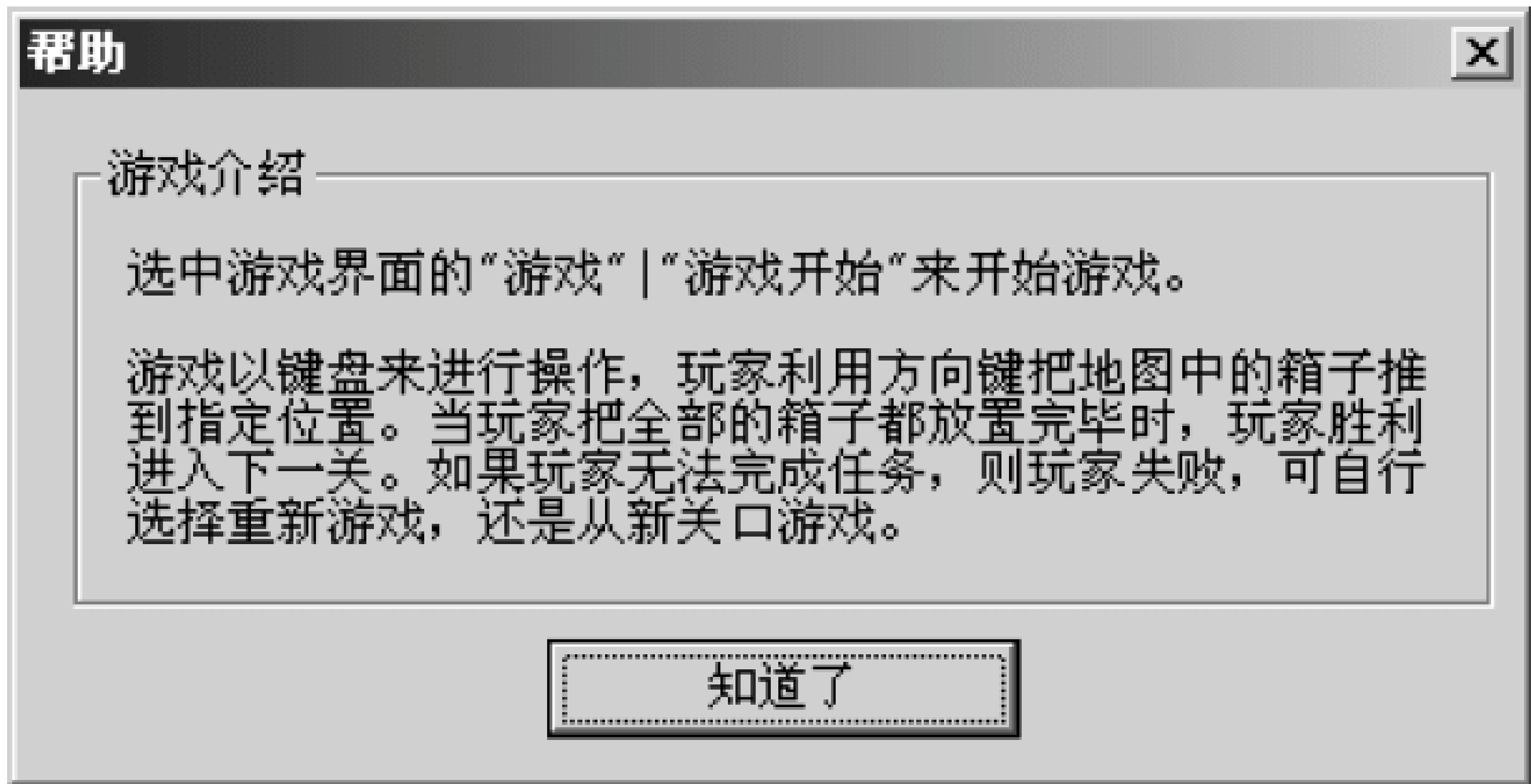


图 16.6　“帮助”对话框

代码 16.2　CHelpDlg 对话框类声明

```
01  #if !defined(AFX_HELPDLG_H__)
02  #define AFX_HELPDLG_H__
03
04  // HelpDlg.h CHelpDlg 类声明头文件
05
06
07  ////////////////////////////////////////////////////////////////////
08  // CHelpDlg 对话框类
09
10  class CHelpDlg : public CDialog              //公共继承于 CDialog 类
11  {
12  public:
13      CHelpDlg(CWnd* pParent = NULL);          //构造函数
14
15  //对话框资源
16      enum { IDD = IDD_HELP };                 //加载资源
17
18  //重载函数
19      protected:
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                     //单击“确定”按钮响应函数声明
25      DECLARE_MESSAGE_MAP()
26  };
27
28  #endif
```

（3）CHelpDlg 对话框类的实现，需要实现对话框类的构造函数、析构函数和“知道了”按钮响应函数，其代码如代码 16.3 所示。

代码 16.3　CHelpDlg 对话框类的实现

```
01  // HelpDlg.cpp CHelpDlg 类的实现源文件
02
03
04  #include "stdafx.h"                          //插入头文件
05  #include "Othello.h"
06  #include "HelpDlg.h"                         //插入类声明头文件
07
08  /////////////////////////////////////////////////////////////
09  // CHelpDlg 对话框类实现
10
```

```
11  CHelpDlg::CHelpDlg(CWnd* pParent /*=NULL*/) //构造函数
12      : CDialog(CHelpDlg::IDD, pParent)
13  {
14  }
15
16  void CHelpDlg::DoDataExchange(CDataExchange* pDX)
17  {
18      CDialog::DoDataExchange(pDX);
19  }
20
21  BEGIN_MESSAGE_MAP(CHelpDlg, CDialog)        //消息与函数映射
22  END_MESSAGE_MAP()
23
24  //////////////////////////////////////////////////////////////
25  // CHelpDlg 消息响应函数
26
27  void CHelpDlg::OnOK()                       //单击“知道了”按钮响应函数
28  {
29      CDialog::OnOK();
30  }
```

16.4.3　游戏关口选择对话框的实现

推箱子游戏关口选择对话框的实现，分为如下几个步骤：

（1）创建一个对话框资源，并添加相应的控件，如图 16.7 所示。



图 16.7　关口选择对话框资源

（2）添加 CSelectDlg 类，其中需要包含 IDD_SELECT_DLG 对话框资源和关口编号变量的声明。类的声明如代码 16.4 所示。

代码 16.4　CHeroDlg 类的声明

```
01  #if !defined(AFX_SELECTDLG _H__)
02  #define AFX_SELECTDLG_H__
03
04  //selectDlg.h 头文件
05  //
06  ////////////////////////////////////////////////////////////////
07  // CSelectDlg 对话框类声明
08
09  class CSelectDlg : public CDialog
10  {
11  public:
12      CSelectDlg(CWnd* pParent = NULL);       //构造函数
13
14      enum { IDD = IDD_HERO_LIST };           //对话框资源
```

```
15      UINT        m_level                     //保存关口编号变量
16
17      public:
18      virtual int DoModal();                  //弹出对话框函数声明
19      protected:
20      virtual void DoDataExchange(CDataExchange* pDX);
21
22  protected:
23
24      virtual void OnOK();                    //单击“设置”按钮响应函数声明
25
26      DECLARE_MESSAGE_MAP()
27  };
28
29  #endif
```

（3）CSelectDlg 类的实现中，主要通过把全局的关口编号变量值赋值给局部变量，并更新显示关口编号到编辑框中。在玩家单击“设置”按钮后，又将获得的局部变量值赋值给全局变量，来改变当前游戏关口，同时要对最大值进行判断。其代码如代码 16.5 所示。

代码 16.5　CSelectDlg 类的实现

```
01  // SelectDlg.cpp : 实现源文件
02
03
04  #include "stdafx.h"                 //插入头文件
05  #include "BoxMan.h"
06  #include "SelectDlg.h"
07
08  #ifdef _DEBUG
09  #define new DEBUG_NEW
10  #undef THIS_FILE
11  static char THIS_FILE[] = __FILE__;
12  #endif
13  /*构造函数*/
14  CSelectDlg::CSelectDlg(CWnd* pParent /*=NULL*/)
15      : CDialog(CSelectDlg::IDD, pParent)
16  {
17      m_level = 0;                    //初始化游戏关口(局部变量)
18  }
19
20
21  void CSelectDlg::DoDataExchange(CDataExchange* pDX)
22  {
23      CDialog::DoDataExchange(pDX);
24      DDX_Text(pDX, IDC_LEVEL, m_level); //将变量与编辑框资源连接
25      DDV_MinMaxUInt(pDX, m_level, 1, g_maxlevel);
                                        //设置关口编号的最大最小值
26  }
27
28  BEGIN_MESSAGE_MAP(CSelectDlg, CDialog)
29  END_MESSAGE_MAP()
30
31  BOOL CSelectDlg::OnInitDialog()     //初始化对话框响应函数
32  {
33      CDialog::OnInitDialog();
34
35      m_level = g_level;              //把当前关口编号赋值给局部变量用于显示
```

```
36
37      return TRUE;                          //初始化成功
38  }
39
40  void CSelectDlg::OnOK()                   //设置按钮响应函数
41  {
42      UpdateData(TRUE);                     //更新输入的数据到变量
43      g_level = m_level;                    //将新设置的关口编号给全局关口编号
44      CDialog::OnOK();
45  }
46  int CSelectDlg::DoModal()                 //弹出对话框响应函数
47  {
48      m_level = g_level;                    //把全局变量的值赋值给等级显示
49
50      return CDialog::DoModal();
51  }
```

代码解析：代码第 40 行的 OnOK()函数主要功能是将输入的游戏关口编号更新到 g_level 变量中，完成游戏等级的提升。

16.4.4　游戏播放背景音乐的实现

播放游戏背景音乐，是通过调用 Windows 的 API 函数 sndPlaySound()来实现的。当玩家选择“游戏设置”|“播放音乐”命令时，就播放音乐。相反，如果取消，就停止播放音乐。要实现这个功能，需要如下几个步骤：

（1）在工程文件中，添加 winmm.lib 静态库文件及头文件，参见第 5.4 节。

（2）实现 CMineDlg 类中的 PlayBackMusic()在成员函数，如代码 16.6 所示。

代码 16.6　CMineDlg 类的 PlayBackMusic 成员函数实现

```
01  #include <mmsystem.h>                     //插入系统 API 头文件
02  ...
03  void CBoxManDlg::PlayBackMusic(BOOL bCheck)
04  {
05      //指定文件并播放
06      if(bCheck)
07      {                                 //播放指定音乐文件
08          sndPlaySound("music.wav",SND_ASYNC);
09      }
10      else
11      {                                 //停止播放
12          sndPlaySound(NULL,SND_PURGE);
13      }
14  }
```

16.5　推箱子游戏的核心算法设计与实现

在前面的章节中已经讲解了推箱子游戏的菜单和各种对话框的实现。本节将对推箱子游戏核心算法的设计和实现进行讲解。

16.5.1　地图文件读取模块的设计与实现

地图文件读取模块，主要负责将对地图文件进行读取，并把相应的文件数据转换成地图显示出来。其设计步骤如下：

（1）读取当前文件夹中的地图文件（map.txt）。

（2）判断当前选择关口是否在文件中存在。

（3）如果存在则把当前关口的地图信息放置到地图数组中。

其实现如代码 16.7 所示。

代码 16.7　地图文件读取模块的实现

```
01  ...                                       //省略部分代码
02  void CBoxManDlg::loadMap(int iMissionNum)
03  {
04      CString str;
05      str.Format("[%d]", iMissionNum);      //格式化字符串，即将关口编号变成[X]
06
07       FILE *pFile = fopen("map.txt", "rb");    //打开地图文件
08       if (pFile == NULL)                       //判断打开是否失败
09         return;
10
11      char cTmp[20];                            //创建临时字符数组
12      fgets(cTmp, 20, pFile);                   //读取一行到数组中
13      while (strncmp(cTmp, str, 3) != 0)//一直读取到与当前关口编号相同的地图头
14      {
15          fgets(cTmp, 20, pFile);
16      }
17
18      for (int i = 0; i < 14; i++)              //读取地图数据到地图数组
19          fgets(m_cMap[i], 20, pFile);
20
21      fclose(pFile);
22  }
```

代码解析：代码第 7 行，通过 FILE 文件指针打开指定地图文件。代码第 13 行利用循环语句一直读取到当前关口编号相同的地图头数据。代码第 19 行，将读取到的地图数据载入到 m_cMap 数组中，将图像显示出来。

16.5.2　地图绘制模块的设计与实现

地图绘制模块主要负责将地图数组中的数据绘制成地图图像。其设计分为如下几个步骤。

（1）根据要求，实现不同类型格子的绘制函数。地图数组及地图文件中各字符代表的意思如表 16.10 所示。

表 16.10　地图文件中数据与图像对照

数据	图　像	宏	说　明
0	黑格	MAP_BACK	用于填充图像
1	白墙	MAP_WHITEWALL	用于阻挡主角移动

续表

数据	图　　像	宏	说　　明
2	蓝格	MAP_BLUEWALL	用于表示可移动空间
3	点	MAP_BALL	用于指定箱子放置位置
4	箱子	MAP_BOX	用于表示初始箱子位置
5	变色的箱子	MAP_REDBOX	用于表示这个点位置已经放置箱子
6	主角	MAP_MAN	用于表示游戏主角位置
7	混合的主角	MAP_MANBALL	用于表示主角是站在点上

（2）读取地图数据，根据不同的数据调用不同的绘图函数。

地图绘制模块的实现如代码 16.8 所示。

代码 16.8　地图绘制模块的实现

```
01  void CBoxManDlg::drawMap(CDC *pDC)              //绘地图接口函数
02  {
03      int i, j, x, y;
04      for (i = 0; i < 14; i++){                   //遍历行
05          for (j = 0; j < 16; j++){               //遍历列
06              x = j * 20;
07              y = i * 20;
08              switch (m_cMap[i][j]){              //根据数组中的数据进行绘图
09              case MAP_BACK:                      //0 黑格
10                  drawBackGroup(x, y, pDC);
11                  break;
12              case MAP_WHITEWALL:                 //1 白墙
13                  drawWhiteWall(x, y, pDC);
14                  break;
15              case MAP_BLUEWALL:                  //2 蓝格
16                  drawBlueWall(x, y, pDC);
17                  break;
18              case MAP_BALL:                      //3 点
19                  drawBall(x, y, pDC);
20                  break;
21              case MAP_BOX:                       //4 箱子
22                  drawBox(x, y, pDC);
23                  break;
24              case MAP_REDBOX:                    //5 变色的箱子
25                  drawRedBox(x, y, pDC);
26                  break;
27              case MAP_MAN:                       //6 主角
28                  drawManWall(x, y, pDC);
29                  break;
30              case MAP_MANBALL:                   //7 混合主角
31                  drawManBall(x, y, pDC);
32                  break;
33              }
34          }
35      }
36  }
37  /*绘黑格函数*/
38  void CBoxManDlg::drawBack (int x, int y, CDC *pDC)
39  {
40      COLORREF clr = RGB(0, 0, 0);
```

```
41      pDC->FillSolidRect(x, y, 20, 20, clr);
42  }
43  /*绘白墙函数*/
44  void CBoxManDlg::drawWhiteWall(int x, int y, CDC *pDC)
45  {
46      COLORREF clr1 = RGB(255, 255, 255);     //白色
47      COLORREF clr2 = RGB(48, 48, 48);
48      COLORREF clr3 = RGB(192, 192, 192);
49      pDC->FillSolidRect(x, y, 19, 19, clr1); //填充矩形着色
50      pDC->FillSolidRect(x + 1, y + 1, 19, 19, clr2);
51      pDC->FillSolidRect(x + 1, y + 1, 18, 18, clr3);
52      pDC->MoveTo(x, y + 10);
53      pDC->LineTo(x + 20, y + 10);            //画线
54      pDC->MoveTo(x + 10, y + 10);
55      pDC->LineTo(x + 10, y + 20);
56  }
57
58  void CBoxManDlg::drawBlueWall(int x, int y, CDC *pDC) /*绘蓝格接口函数*/
59  {
60      COLORREF clr = RGB(0, 0, 255);          //蓝色
61      pDC->FillSolidRect(x, y, 20, 20, clr);  //填充矩形着色
62      pDC->MoveTo(x, y + 10);
63      pDC->LineTo(x + 20, y + 10);            //画线
64      pDC->MoveTo(x + 10, y + 10);
65      pDC->LineTo(x + 10, y + 20);
66  }
67  /*绘点接口函数*/
68  void CBoxManDlg::drawBall(int x, int y, CDC *pDC)
69  {
70      COLORREF clr = RGB(0, 0, 255);          //蓝色
71      pDC->FillSolidRect(x, y, 20, 20, clr);  //填充着色
72      pDC->MoveTo(x, y + 10);
73      pDC->LineTo(x + 20, y + 10);            //画线
74      pDC->MoveTo(x + 10, y + 10);
75      pDC->LineTo(x + 10, y + 20);
76      pDC->Ellipse(x, y, x + 20, y + 20);     //画圆
77      pDC->Ellipse(x + 5, y + 5, x + 15, y + 15);
78  }
79  /*绘箱子接口函数*/
80  void CBoxManDlg::drawBox(int x, int y, CDC *pDC)
81  {
82      COLORREF clr = RGB(255, 255, 0);        //黄色
83      pDC->FillSolidRect(x, y, 20, 20, clr);  //填充矩形着色
84      COLORREF clr2 = RGB(255, 192, 0);
85      pDC->FillSolidRect(x + 2, y + 2, 16, 16, clr2);
86      COLORREF clr3 = RGB(0, 0, 0);
87      pDC->SetPixel(x + 3, y + 3, clr3);      //画点颜色
88      pDC->SetPixel(x + 17, y + 3, clr3);
89      pDC->SetPixel(x + 3, y + 17, clr3);
90      pDC->SetPixel(x + 17, y + 17, clr3);
91  }
92  /*绘变色箱子接口函数*/
93  void CBoxManDlg::drawRedBox(int x, int y, CDC *pDC)
94  {
95      COLORREF clr = RGB(255, 255, 0);        //黄色
96      pDC->FillSolidRect(x, y, 20, 20, clr);  //填充矩形着色
97      COLORREF clr2 = RGB(255, 0, 0);         //绿色
```

```
98      pDC->FillSolidRect(x + 2, y + 2, 16, 16, clr2);
99      COLORREF clr3 = RGB(0, 0, 0);
100     pDC->SetPixel(x + 3, y + 3, clr3);
101     pDC->SetPixel(x + 17, y + 3, clr3);
102     pDC->SetPixel(x + 3, y + 17, clr3);
103     pDC->SetPixel(x + 17, y + 17, clr3);
104 }
105 /*绘制主角人物接口函数*/
106 void CBoxManDlg::drawMan(int x, int y, CDC *pDC)
107 {
108     COLORREF clr = RGB(0, 0, 255);                //蓝色格子
109     pDC->FillSolidRect(x, y, 20, 20, clr);
110     pDC->MoveTo(x, y + 10);
111     pDC->LineTo(x + 20, y + 10);
112     pDC->MoveTo(x + 10, y + 10);
113     pDC->LineTo(x + 10, y + 20);
114
115     pDC->Ellipse(x + 6, y + 2, x + 14, y + 10); //主角的头
116     pDC->MoveTo(x + 2, y + 11);                   //主角的手
117     pDC->LineTo(x + 18, y + 11);
118     pDC->MoveTo(x + 10, y + 10);                  //绘制主角的身体
119     pDC->LineTo(x + 10, y + 12);
120     pDC->MoveTo(x + 2, y + 20);                   //绘制主角的脚
121     pDC->LineTo(x + 10, y + 12);
122     pDC->LineTo(x + 18, y +20);
123 }
124 /*绘制主角人物在点上的接口函数*/
125 void CBoxManDlg::drawManBall(int x, int y, CDC *pDC)
126 {
127     COLORREF clr = RGB(0, 0, 255);                //点
128     pDC->FillSolidRect(x, y, 20, 20, clr);
129     pDC->MoveTo(x, y + 10);
130     pDC->LineTo(x + 20, y + 10);
131     pDC->MoveTo(x + 10, y + 10);
132     pDC->LineTo(x + 10, y + 20);
133     pDC->Ellipse(x, y, x + 20, y + 20);
134     pDC->Ellipse(x + 5, y + 5, x + 15, y + 15);
135
136     pDC->Ellipse(x + 6, y + 2, x + 14, y + 10); //主角的头
137     pDC->MoveTo(x + 2, y + 11);                   //主角的手
138     pDC->LineTo(x + 18, y + 11);
139     pDC->MoveTo(x + 10, y + 10);                  //主角的身体
140     pDC->LineTo(x + 10, y + 12);
141     pDC->MoveTo(x + 2, y + 20);                   //主角的脚
142     pDC->LineTo(x + 10, y + 12);
143     pDC->LineTo(x + 18, y +20);
144 }
```

代码解析：代码第 8 行，先遍历当前地图数组中的数据，然后根据其不同数据类型调用不同的显示函数，显示相应的图片。后面的代码都是调用绘图函数进行图像的绘制工作的。

16.5.3　键盘操作模块的设计与实现

键盘操作模块主要负责接收玩家键盘输入并进行箱子移动等处理。其设计比较简单，

只需要通过如下几步即可实现。

（1）通过截获当前窗口中键盘按下消息来判断玩家按下的按键。

（2）根据玩家按下的按键把主角的相关坐标进行加减。例如，当玩家按下方向键“右”时，把主角的 X 坐标增加，反之，把主角的 X 坐标减少。当玩家按下方向键“上”时，把主角的 Y 坐标减少，反之，把主角的 Y 坐标增加。

（3）把增加后的数据与地图数组中的数据进行比较，只要不是白墙就可以移动，如果是箱子还需要将箱子的坐标进行移动。

（4）但要注意，要判断箱子移动后的坐标是否可以移动。

键盘操作模块的实现函数，如代码 16.9 所示。

代码 16.9　键盘操作模块的函数实现

```
01  /*消息截获函数*/
02  BOOL CBoxManDlg:: PreTranslateMessage(MSG* pMsg)
03  {
04      if(pMsg->message == WM_KEYDOWN) {
05          updateMap(pMsg->wParam);        //根据按下的按键进行判断
06          Invalidate(false);
07          if (isFinish())                 //调用规则模块函数，判断是否胜利
08          {
09              AfxMessageBox("您获得胜利,将进入下一关!");
10              g_level = g_level +1;       //胜利后将当前关口增加 1
11              if(g_level > g_maxlevel)
12                  g_level = 1;            //如果大于最大关口编号，重新开始
13              loadMap(g_level);           //加载指定关口编号地图
14          m_ptManPosition = getManPosition();
15      }
16      return CDialog::PreTranslateMessage(pMsg);
17  }
18  /*更新地图函数*/
19  void CBoxManDlg::updateMap(UINT nChar)
20  {
21      int x1, y1, x2, y2, x3, y3;
22
23      x1 = m_ptManPosition.x;                 //得到主角的 X 坐标
24      y1 = m_ptManPosition.y;                 //得到主角的 Y 坐标
25
26      switch (nChar)                          //判断按下的按钮
27      {
28      case VK_UP:                             //方向键上
29          x2 = x1;
30          y2 = y1 - 1;
31          x3 = x1;
32          y3 = y1 - 2;
33          ReDrawMap(x1, y1, x2, y2, x3, y3);  //重新绘制地图
34          break;
35      case VK_DOWN:                           //方向键下
36          x2 = x1;
37          y2 = y1 + 1;
38          x3 = x1;
39          y3 = y1 + 2;
40          ReDrawMap(x1, y1, x2, y2, x3, y3);
41          break;
42      case VK_LEFT:                           //方向键左
```

```
43          x2 = x1 - 1;
44          y2 = y1;
45          x3 = x1 - 2;
46          y3 = y1;
47          ReDrawMap(x1, y1, x2, y2, x3, y3);
48          break;
49      case VK_RIGHT:                          //方向键右
50          x2 = x1 + 1;
51          y2 = y1;
52          x3 = x1 + 2;
53          y3 = y1;
54          ReDrawMap(x1, y1, x2, y2, x3, y3);
55          break;
56       default:
57           break;
58      }
59  }
60  /*根据新的数据，重绘地图*/
61  void CBoxManDlg::ReDrawMap(int x1, int y1, int x2, int y2,int x3,int y3)
62  {
63      switch (m_cMap[y2][x2])
64      {
65      case MAP_BACKG:                   //如果移动到黑格，说明地图文件有问题
66          MessageBox("地图文件有问题");
67          break;
68      case MAP_WHITEWALL:                     //如果遇到白墙，不做移动
69          break;
70      case MAP_BLUEWALL:                      //如果移动后的位置是蓝格
71          m_cMap[y2][x2] = MAP_MANWALL;       //移动后的位置为人和墙会成状态
72          if (m_cMap[y1][x1] == MAP_MANWALL)  //*恢复原来格子的状态*/
73              m_cMap[y1][x1] = MAP_BLUEWALL;
74          else if (m_cMap[y1][x1] == MAP_MANBALL)
75              m_cMap[y1][x1] = MAP_BALL;
76          m_ptManPosition.x = x2;             //更新主角坐标
77          m_ptManPosition.y = y2;
78          break;
79      case MAP_BALL:                          //移动后的位置是点
80           m_cMap[y2][x2] = MAP_MANBALL;      //改变移动后格子的状态
81          if (m_cMap[y1][x1] == MAP_MANWALL)  /*恢复移动前的格子状态*/
82              m_cMap[y1][x1] = MAP_BLUEWALL;
83          else if (m_cMap[y1][x1] == MAP_MANBALL)
84              m_cMap[y1][x1] = MAP_BALL;
85          m_ptManPosition.x = x2;
86          m_ptManPosition.y = y2;
87          break;
88      case MAP_BOX:                           //移动后的位置有箱子
89          if (m_cMap[y3][x3] == MAP_BALL)     //判断箱子是否可以移动
90          {/*箱子可以移动，设置移动后的格子状态*/
91              m_cMap[y3][x3] = MAP_REDBOX;
92              m_cMap[y2][x2] = MAP_MANWALL;
93              if (m_cMap[y1][x1] == MAP_MANWALL)
94                  m_cMap[y1][x1] = MAP_BLUEWALL;
95              else if (m_cMap[y1][x1] == MAP_MANBALL)
96                  m_cMap[y1][x1] = MAP_BALL;
97              m_ptManPosition.x = x2;
98              m_ptManPosition.y = y2;
99          }
100         else if (m_cMap[y3][x3] == MAP_BLUEWALL)
```

```
101         {/*是蓝格，直接移动箱子，并恢复当前格子的状态*/
102             m_cMap[y3][x3] = MAP_BOX;
103             m_cMap[y2][x2] = MAP_MANWALL;
104             if (m_cMap[y1][x1] == MAP_MANWALL)
105                 m_cMap[y1][x1] = MAP_BLUEWALL;
106             else if (m_cMap[y1][x1] == MAP_MANBALL)
107                 m_cMap[y1][x1] = MAP_BALL;
108             m_ptManPosition.x = x2;
109             m_ptManPosition.y = y2;
110         }
111         break;
112     case MAP_REDBOX:                                //移动后的位置是变色的箱子
113         if (m_cMap[y3][x3] == MAP_BALL)       //判断箱子是否可以移动
114         {
115             m_cMap[y3][x3] = MAP_REDBOX;
116             m_cMap[y2][x2] = MAP_MANBALL;
117             if (m_cMap[y1][x1] == MAP_MANWALL)
118                 m_cMap[y1][x1] = MAP_BLUEWALL;
119             else if (m_cMap[y1][x1] == MAP_MANBALL)
120                 m_cMap[y1][x1] = MAP_BALL;
121             m_ptManPosition.x = x2;
122             m_ptManPosition.y = y2;
123         }
124         else if (m_cMap[y3][x3] == MAP_BLUEWALL)
125         {/*判断箱子是否可以移动，并设置移动后的格子状态*/
126             m_cMap[y3][x3] = MAP_BOX;
127             m_cMap[y2][x2] = MAP_MANBALL;
128             if (m_cMap[y1][x1] == MAP_MANWALL)
129                 m_cMap[y1][x1] = MAP_BLUEWALL;
130             else if (m_cMap[y1][x1] == MAP_MANBALL)
131                 m_cMap[y1][x1] = MAP_BALL;
132             m_ptManPosition.x = x2;
133             m_ptManPosition.y = y2;
134         }
135         break;
136     case MAP_MAN:                         //移动后的格子有主角存在，说明地图有问题
137         MessageBox("地图文件有问题");
138         break;
139     case MAP_MANBALL:                     //移动后的格子有主角存在，说明地图有问题
140         MessageBox("地图文件有问题");
141         break;
142     }
143 }
```

代码解析：代码第 4 行是利用截获当前窗口中键盘按下消息来判断玩家按下的按键。根据玩家按下的按键把主角的相关坐标进行加减。如果移动后根据规则游戏结束，则进入下一关，并重新加载地图；否则继续游戏。

注意：如果在 CDialog 类中直接使用响应键盘消息的 OnKeyDown()函数，CDialog 是不响应 KeyDown 消息的，相应的解决方法是使用 PreTranslateMessage 函数来截获其中的 KeyDown 消息。

16.5.4　游戏规则模块的设计与实现

游戏规则模块：主要负责游戏规则的判断。其设计方法比较简单，需要在每次玩家移

动主角后，对当前地图数组进行判断。如果当前地图中存在单独的“点”或者“主角站在点”上两种情况，就说明玩家未胜利；反之就是胜利。

游戏规则模块的函数实现，如代码 16.10 所示。

代码 16.10　游戏规则模块的函数实现

```
01  BOOL CBoxManDlg::isFinish()
02  {
03      for (int i = 0; i < 14; i++)              //遍历所有列
04      {
05          for (int j = 0; j < 16; j++)          //遍历所有行
06          {
07              if (m_cMap[i][j] == MAP_BALL || m_cMap[i][j] == MAP_MANBALL)
08              {
09                  return FALSE;                 //有点或者主角站在点上存在
10              }
11          }
12      }
13      return TRUE;                              //游戏结束
14  }
```

16.5.5　主对话框的设计与实现

在推箱子游戏中，主对话框的设计比较简单，主要有如下几部分。

1. 游戏初始化处理函数

主要负责对推箱子游戏进行初始化，例如调用加载地图函数、设置最大关口编号、初始化行进步数等。

2. 菜单初始化处理函数

主要是对菜单中的“背景音乐”栏在游戏开始时，设置为“未被选中”状态，以及其他一些初始化功能。

3. 绘图函数处理

主要通过调用地图绘制函数，把当前设备的指针赋值过去，这样地图绘制函数才能对当前的窗口进行地图绘制。

主对话框的部分实现，如代码 16.11 所示。

代码 16.11　主对话框的部分函数实现

```
01  /*绘图成员函数*/
02  void CBoxManDlg::OnPaint()
03  {
04      CPaintDC dc(this);                  //得到当前窗口设置
05      drawMap(&dc);                       //调用地图绘制函数
06      CDialog::OnPaint();                 //调用基类绘图函数
07  }
08  /*游戏初始化函数*/
```

```
09  void CBoxManDlg::GameStart()
10  {
11      loadMap(g_level);                       //加载指定编号地图
12      m_ptManPosition = getManPosition();//得到主角坐标
13      passtime = 0;                           //初始化已走步数
14      Invalidate();
15  }
16  /*菜单初始化*/
17  void CBoxManDlg::InitMenu()
18  {
19      CWnd*   pMain   =   AfxGetMainWnd();
20      CMenu*   pMenu   =   pMain->GetMenu();
21      pMenu->CheckMenuItem(IDR_PLAY_MUSIC,
22          MF_BYCOMMAND | MF_UNCHECKED);
23  }
```

代码解析：代码第 9 行是游戏初始化函数，主要是完成加载指定编号地图，并得到游戏主角坐标的功能。

16.6　推箱子游戏的整合测试

在本节中，笔者将根据前面的测试用例文档进行推箱子游戏的整合测试。在这里，笔者只对其中几个主要的测试用例进行演示。

16.6.1　主菜单和界面显示功能的测试演示

这个测试用例主要是测试游戏的菜单和界面显示是否成功，根据测试用例文档，其测试步骤如下所示。

（1）运行推箱子程序，选中其中的.exe 图标，如图 16.8 所示。

（2）程序启动后，其菜单及主界面如图 16.9 所示。



图 16.8　运行推箱子程序

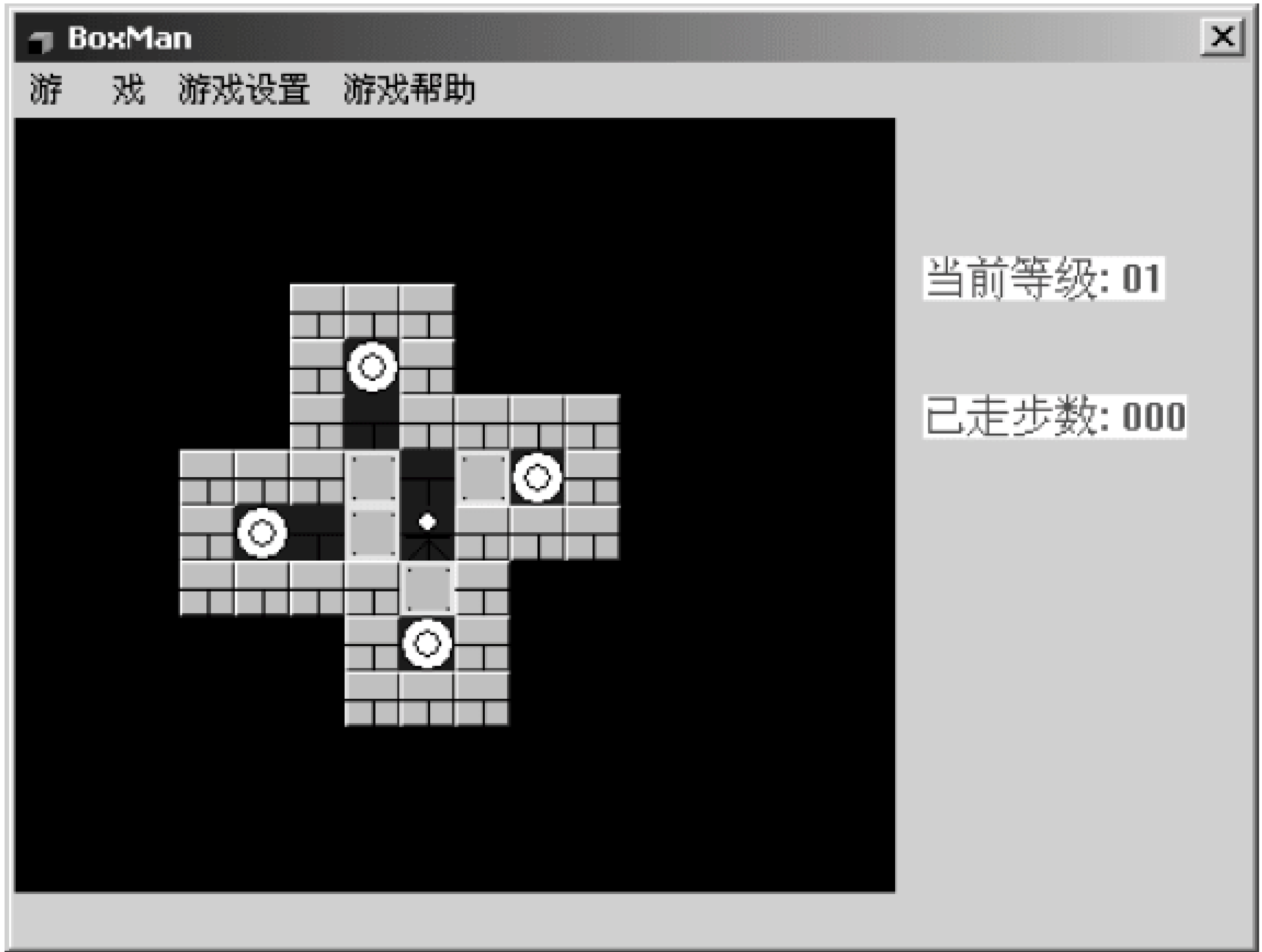


图 16.9　推箱子游戏主界面及菜单

判断结果：游戏的菜单和界面显示成功。填写测试用例编号 1.7.1 结果为“通过”。

16.6.2　键盘操作功能的测试演示

键盘操作功能测试，根据测试用例文档，其测试步骤如下所述（这里只对其中的方向键“下”进行测试演示）。

（1）游戏开始后的主角位置，如图 16.10 所示。

（2）单击键盘上的“下”方向按键。

（3）查看主角是否向左移动，如图 16.11 所示。



图 16.10　游戏开始后的主角位置

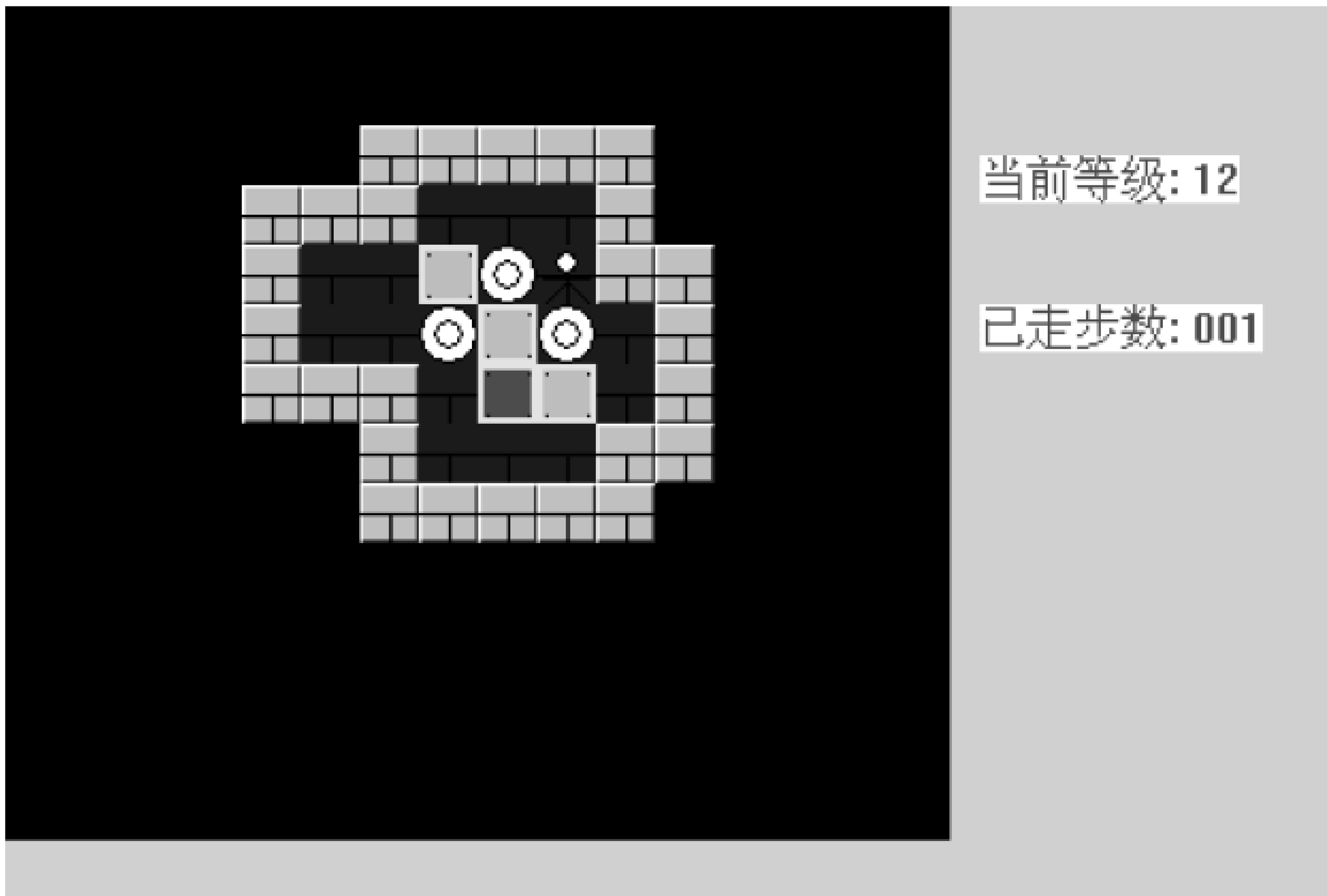


图 16.11　单击“下”方向键后

判断结果：键盘操作功能测试成功。填写测试用例编号 1.7.2 结果为“通过”。

16.6.3　箱子放置到指定位置时变色显示功能的测试演示

测试将箱子放置到指定位置时，箱子会变色的功能。根据测试用例文档，其测试步骤如下所述。

（1）推箱子游戏已经开始，如图 16.12 所示。

（2）推动箱子到指定位置（地图中的点位置）后，显示如图 16.13 所示。

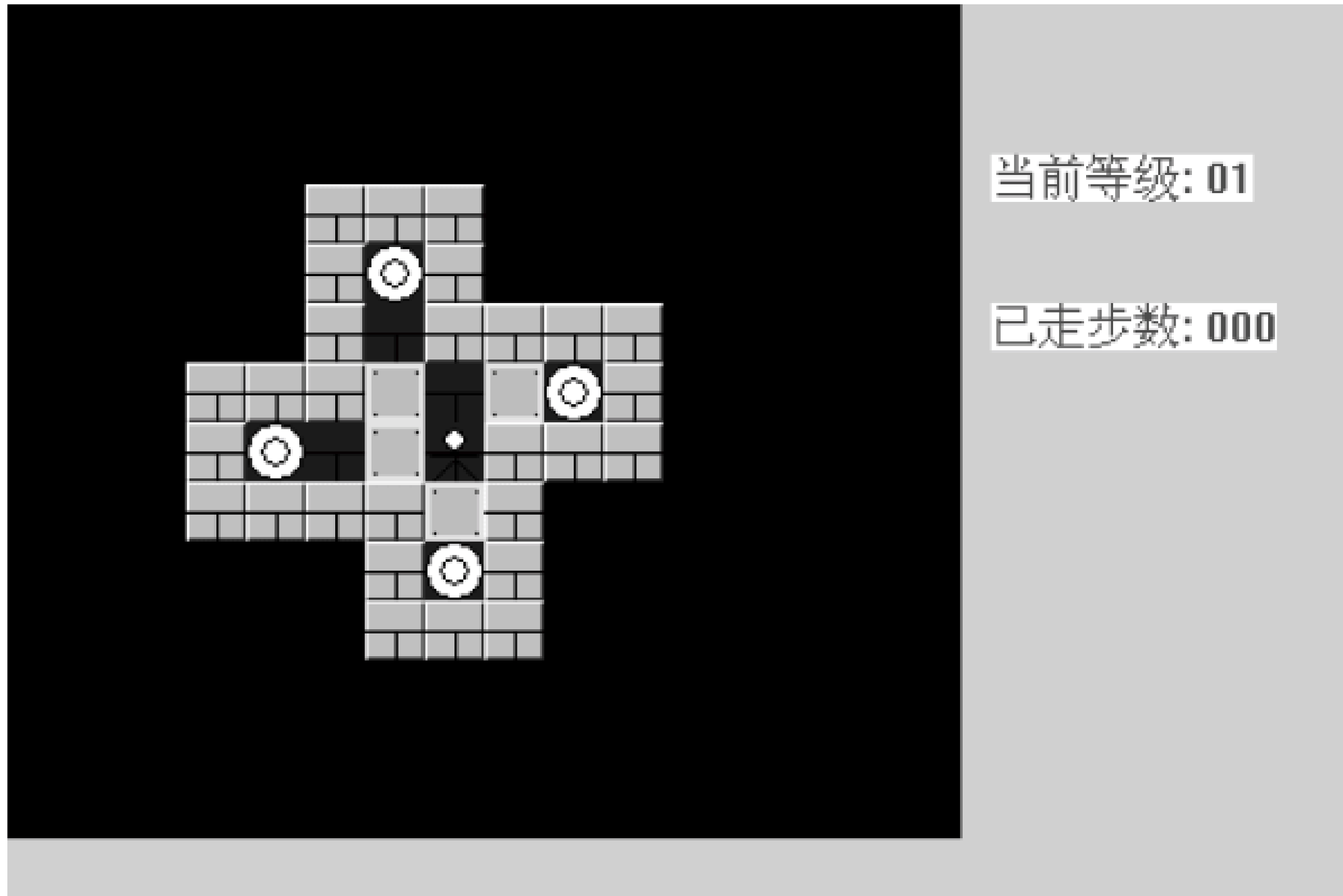


图 16.12　初始状态

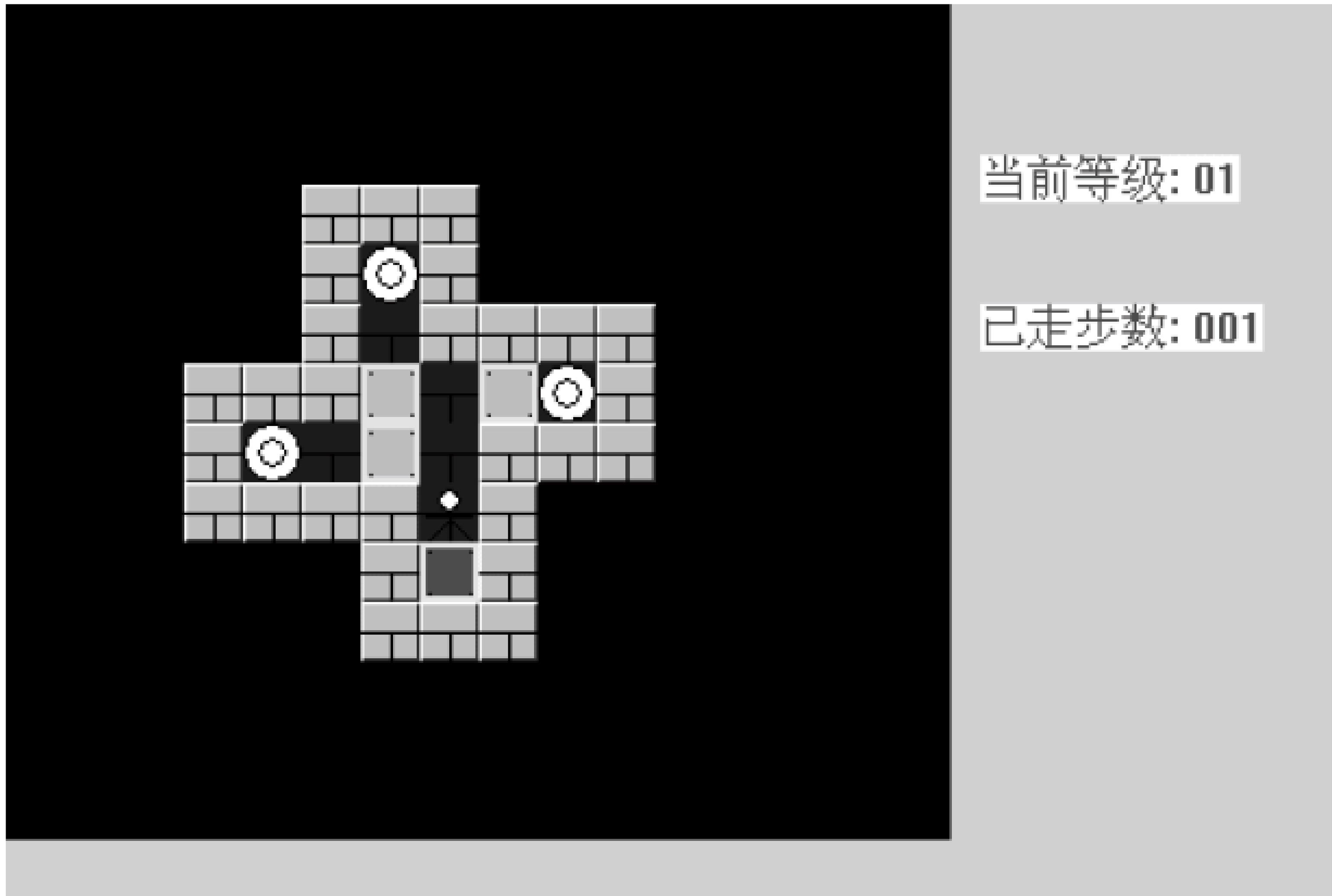


图 16.13　移动箱子到点上

判断结果：通过比较，将箱子放置到指定位置后，箱子变色功能测试正确。填写测试用例编号 1.7.3 结果为“通过”。

16.6.4　支持地图扩展功能的测试演示

测试推箱子游戏是否支持地图扩展功能。根据测试用例文档，其测试步骤如下所述。

（1）查看第一关地图文件，其内容如图 16.14 所示。

（2）与游戏显示的地图对照（如图 16.12）。

（3）修改其中主角下方的箱子数据（4）为白墙类型（1），如图 16.15 所示。

```
[1]
0000000000000000
0000000000000000
0000000000000000
0000011100000000
0000013100000000
0000012111100000
0001114243100000
0001324611100000
0001111410000000
0000001310000000
0000001110000000
0000000000000000
0000000000000000
0000000000000000
```

图 16.14　第一关地图文件内容

```
[1]
0000000000000000
0000000000000000
0000000000000000
0000011100000000
0000013100000000
0000012111100000
0001114243100000
0001324611100000
0001111110000000
0000001310000000
0000001110000000
0000000000000000
0000000000000000
0000000000000000
```

图 16.15　修改后的文件内容

（4）重新进行第一关游戏，地图显示如图 16.16 所示。

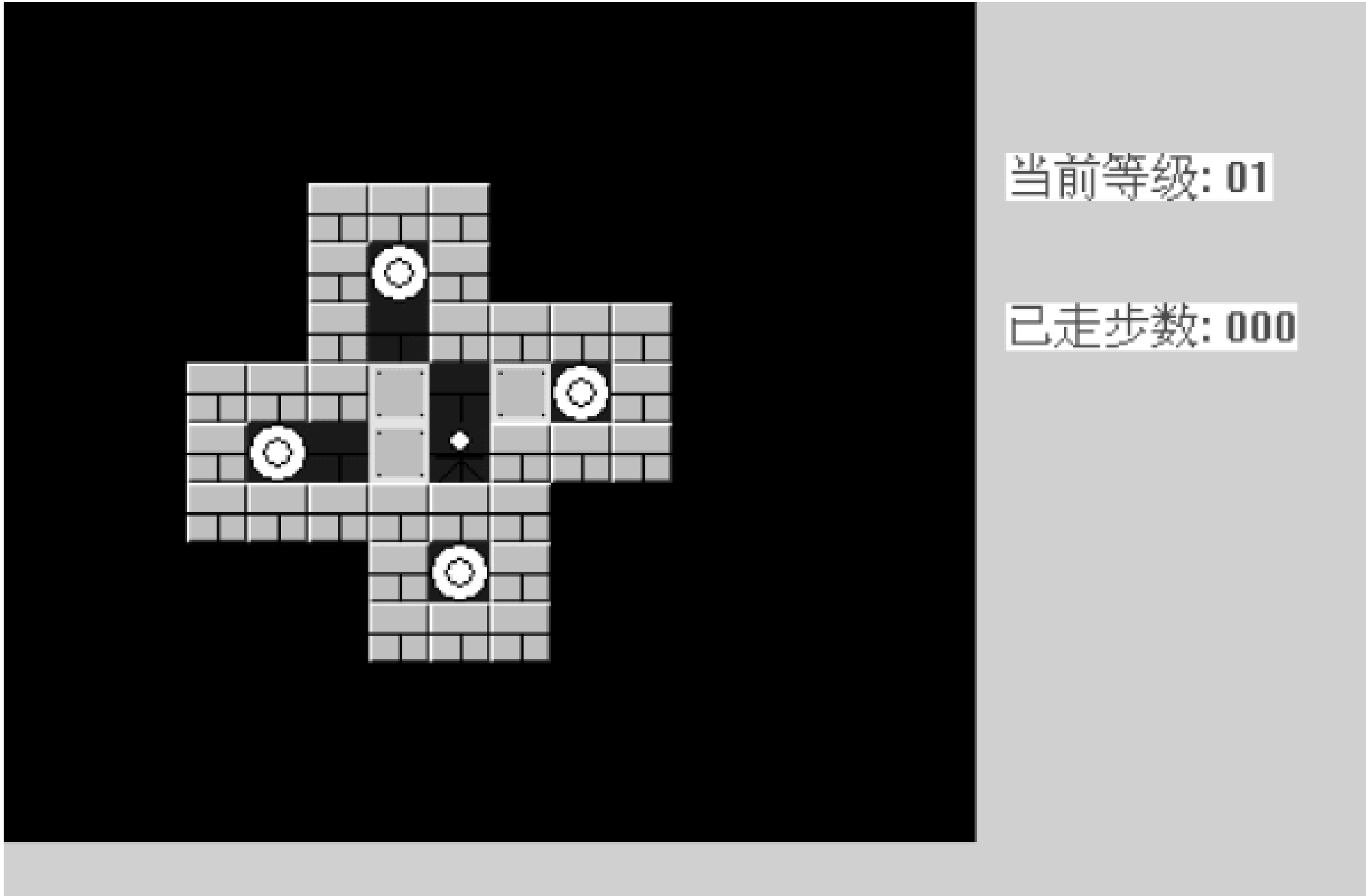


图 16.16　修改后的游戏地图

判断结果：推箱子游戏中地图扩展功能正确。填写测试用例编号 1.7.4 结果为“通过”。

16.6.5　游戏胜负判断功能的测试演示

测试推箱子游戏中游戏胜负判断功能。由于推箱子游戏的特殊性，所以只测试其在玩家胜利时，能否进入下一关。根据测试用例文档，其测试步骤如下所述。

（1）游戏开始后，玩家把将要推的箱子全部推到指定位置，只剩余最后一个箱子，如图 16.17 所示。

（2）将最后一个箱子推到指定位置。显示结果如图 16.18 所示。

判断结果：推箱子游戏中游戏胜负判断功能正确。填写测试用例编号 1.7.5 结果为“通过”。



图 16.17　剩余最后一个箱子



图 16.18　全部箱子推到指定位置后

16.6.6　游戏帮助功能的测试演示

测试推箱子游戏是否有帮助提示功能。根据测试用例文档，其测试步骤如下所述。

（1）选择“帮助”|“帮助”命令，如图 16.19 所示。

（2）游戏中弹出“帮助”对话框，如图 16.20 所示。

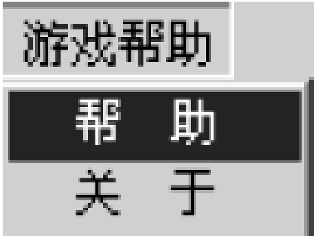


图 16.19　选择“游戏帮助”|“帮助”命令

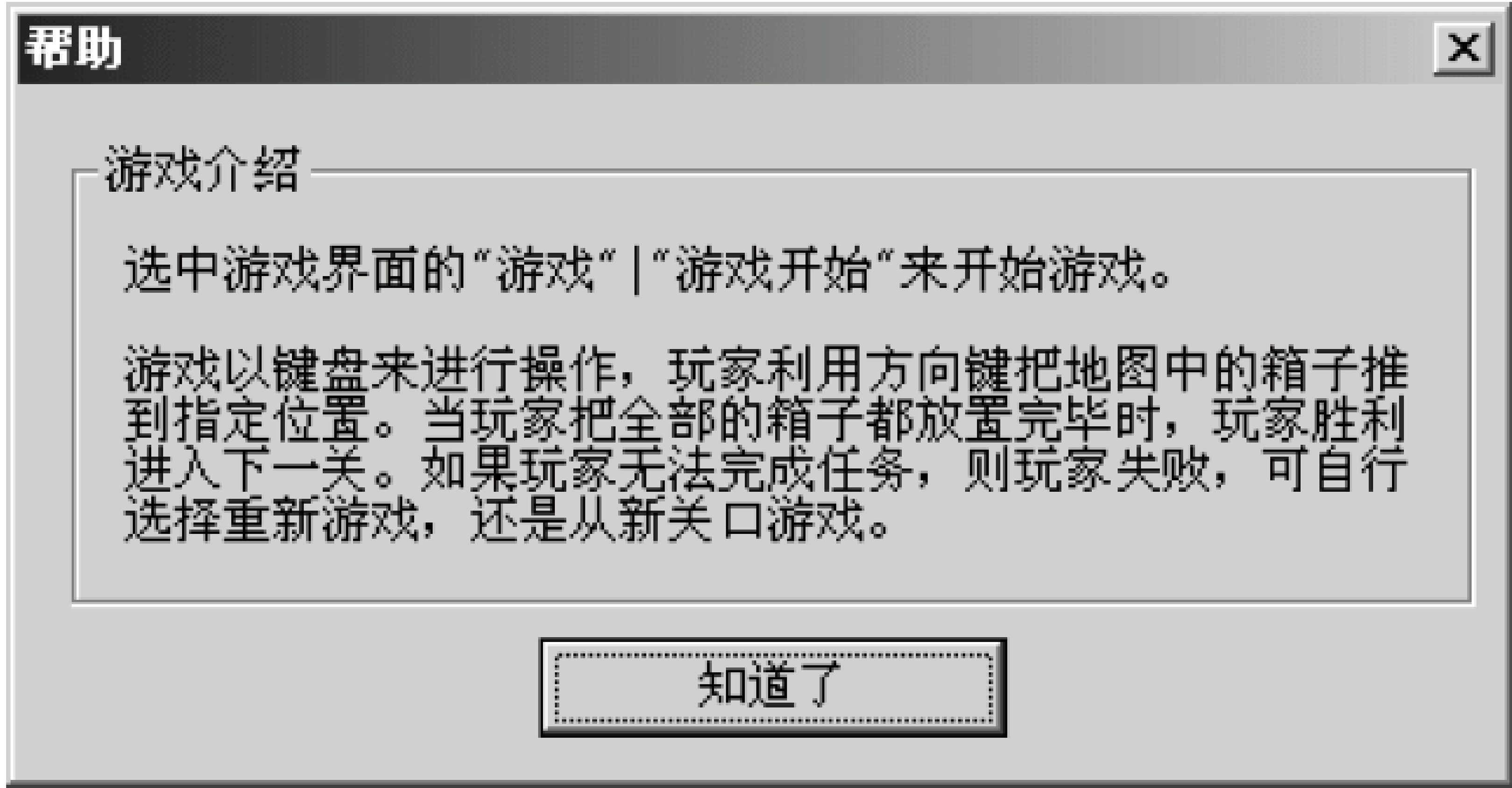


图 16.20　“帮助”对话框

判断结果：推箱子游戏帮助提示是正确的。填写测试用例编号 1.7.8 结果为“通过”。

16.7　总　　结

通过对第 16 章推箱子游戏实例项目开发的学习，相信各位读者已经对项目开发文档的编写比较熟悉了。虽然文档的编写比较枯燥、烦琐和格式化，但正是因为有了各种项目文档，才保证项目能够顺利进行，达到目标。

同时，读者要学好推箱子游戏中包括绘图模块、键盘操作模块、地图文件读取模块等核心算法的实现，最终能够自己开发一款推箱子游戏。

最后，衷心地希望每一位读者都能够掌握好本书的相关知识，在以后的工作中能有所收获和突破。